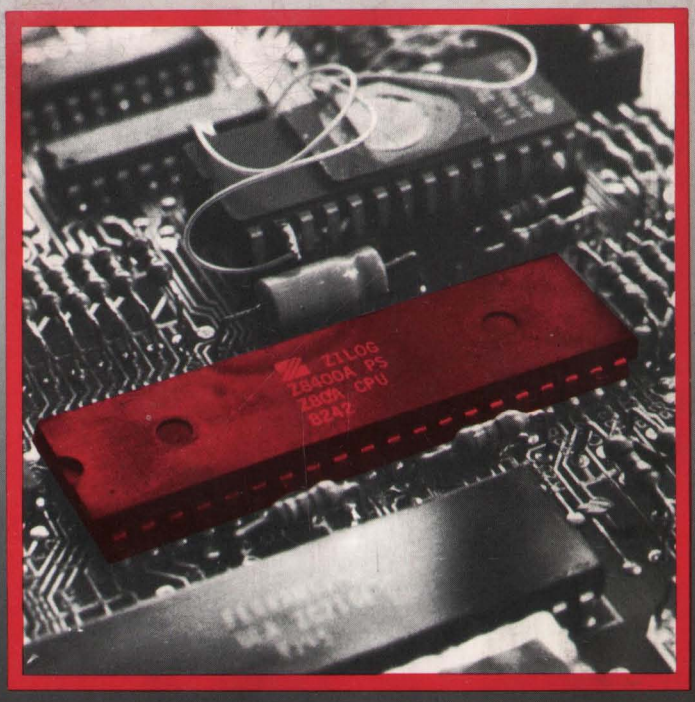
Jochen Merz

Maschinencode-Fibel für den ZX 81

und Z80-Processor



profisoft

Jochen Merz

Maschinencode-Fibel für den ZX 81

und Z 80-Processor

profisoft

ISBN 3-923985-00-2

Herausgeber: Profisoft GmbH, 4500 Osnabrück, Sutthauser Straße 50/52
Alle Rechte vorbehalten.
Vervielfältigungen ohne Genehmigung des Herausgebers nicht gestattet.
Druck und Gesamtherstellung: A & C Welchert GmbH, 4930 Detmold

Vorwort

Sicherlich haben Sie sich als ZX-81 Anwender schon eingehender mit diesem, von den Abmessungen her kleinen, aber in der Leistung großen Microcomputer beschäftigt.

Aller Einstieg in die Microprozessortechnik ist schwer; besonders für Newcomer. Neben dem neuen Stoff muß auch noch das für Deutsche oft seltsam anmutende „computer english" erlernt werden.

Dies hat auch der Autor dieses Büchleins, Jochen Merz, feststellen müssen. Er merkte bald, daß die auf dem Markt erhältlichen Bücher über den Z80 Prozessor entweder theoretische Abhandlungen darstellten, oder aber so allgemein verfaßt waren, daß ein konkretes Arbeiten nicht möglich war.

Wir von Profisoft freuen uns deshalb umso mehr, Ihnen ein völlig neues „Einsteiger-Buch" vorstellen zu können. Obwohl auch die Fibel genug Anlaß zur Kritik bieten mag (Umfang, weitergehende Beispiele) hoffen wir dennoch, Ihnen hiermit eine erste Hilfe für den Einstieg in die Maschinensprache an die Hand zu geben, die leichtverständlich ist und Erfolgserlebnisse produziert.

Osnabrück, im April 1983

das Team von Profisoft

Inhalt

1. Der Z80-Mikroprozessor 1
1.1 Die Datenbusse 2
1.2 Die Kontrolleinheit 2
1.3 Die Recheneinheit 2
1.4 Das Befehlsregister 3
1.5 Der Programmzähler 3
1.6 Die Hauptregister 3
1.6.1 Das A-Register 3
1.6.2 Das F-Register 3
1.6.3 Das HL-Registerpaar 3
1.6.4 Die Registerpaare BC und DE 4
1.6.5 Das IX-Registerpaar 4
1.6.6 Das IY-Registerpaar 4
1.6.7 Der Stackpointer 4
1.6.8 Das R-Register 4
1.6.9 Das I-Register 4

2. Grundlagen der Maschinencodebefehle 5
2.1 Rechnen mit binären Zahlen 5

3. Die Z80-Maschinencodebefehle 7
3.1 Die Flags 7
3.1.1 Das Zero Flag 7
3.1.2 Das Sign Flag 7
3.1.3 Das Parity/Overflow Flag 7
3.1.4 Das Carry Flag 8
3.2.1 NOP 8
3.2.2 RET 8
3.2.3 Ladebefehle 8
3.2.4 Registeraustausch 13
3.2.5 Rechenoperationen 14
3.2.6 Vergleichsoperatoren 18
3.2.7 Logische Operatoren 18
3.2.8 Sprungbefehle 20
3.2.9 Unterprogramm-Befehle 23
3.2.10 Stapeloperationen 24
3.2.11 Rotations- und Schiebebefehle 25
3.2.12 Einzelbitverarbeitung 27
3.2.13 Blockverschiebebefehle 28
3.2.14 Blocksuchbefehle 29
3.2.15 Transfer-Befehle 30
3.2.16 Interrupt-Befehle 31
3.2.17 Sonstige Befehle für den Akkumulator 32

4. Maschinencode in BASIC 33

5. Programmbeispiele 34
5.1 SCROLL 34
5.2 CLS 34
5.3 RAMTOP-Verschiebung 35

6. Listen, Tabellen 37
6.1 Einsprungadressen des ROM (alphabetisch) 37
6.2 Einsprungadressen des ROM (numerisch) 41
6.3 Umwandlungstabelle Dezimal/Hexadez/Binär/Zweierkomplement 45
6.4 Umwandlungstabelle Hexadezimal/Dezimal (höherwert. Byte) 46
6.5 Z80-Befehlssatz (numerisch) 48
6.6 Index-Befehle 52

1. Der Z80-Mikroprozessor

Der ZX81 besitzt als wichtigsten Chip den Z80-Mikroprozessor. Ein Mikroprozessor ist ein Siliziumchip, der, um mit anderen Chips in Verbindung zu treten, Ein- und Ausgabeleitungen besitzt. Über Eingabeleitungen werden elektrische Impulse in den Z80 gebracht, über Ausgabeleitungen gibt der Z80 Impulse an andere Chips weiter. Intern ist der Z80 so kompliziert aufgebaut, daß man jede einzelne Funktion kaum besprechen kann. Man kann ihn deshalb grob in fünf Gebiete aufteilen.
Abb. 1 zeigt einen dementsprechenden Aufbau des Z80.

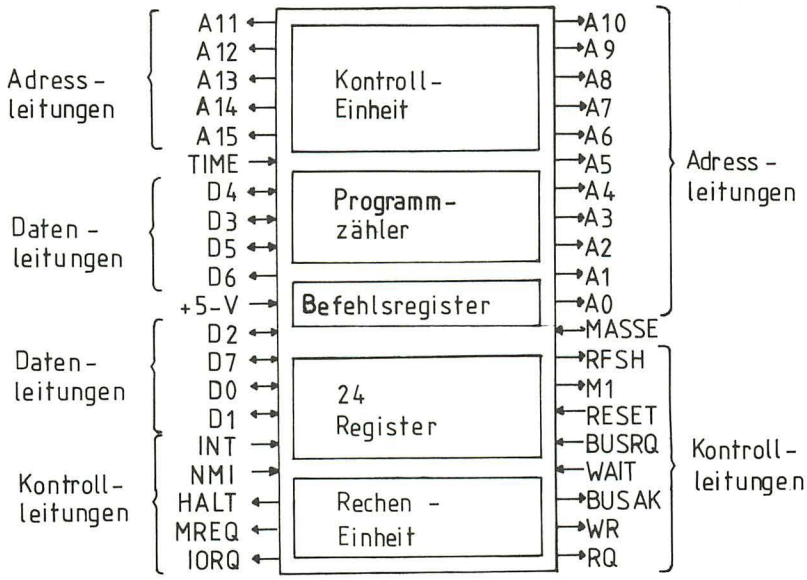


Abb. 1

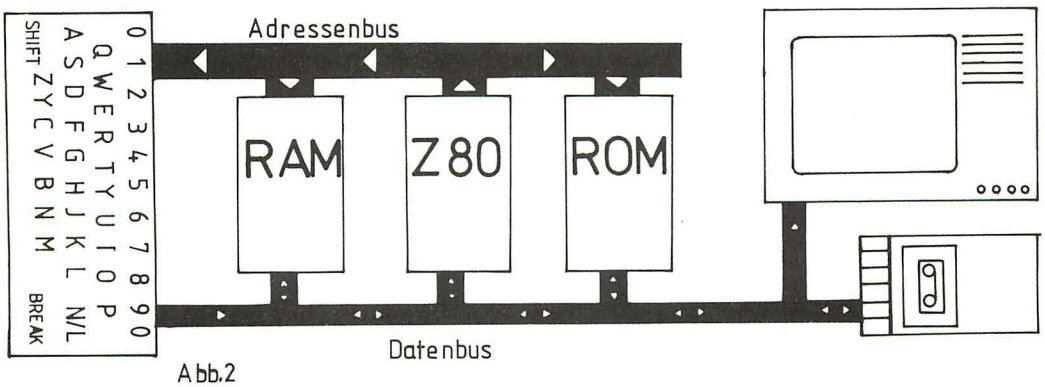


Abb.2

Der Z80 ist wie ein Computer konzipiert, denn er ist in der Lage, ein Programm zu interpretieren und abzuarbeiten. Nur ist das Programm nicht in BASIC geschrieben, sondern in einer Reihe von Maschinencode-befehlen. Dieses ganze Maschinencodeprogramm, welches im 8K-ROM steht, ist das Programm, mit dem Computer z.B. den Bildschirm generiert, BASIC-Befehle abarbeitet oder auf Tastendrücke entsprechend reagiert. Dieses Maschinencodeprogramm wird ab jetzt 'Monitor' genannt.
Der Monitor des ZX81 liegt so, daß der Z80 sofort mit dem Abarbeiten beginnt, wenn der Strom eingeschaltet wird.

1.1 Die Datenbusse

Der Z80 kann nicht alleine arbeiten und muß daher mit anderen Bauteilen in Verbindung stehen.
Es gibt drei verschiedene Arten von Verbindungen:

- Datenleitungen
- Adressenleitungen
- Kontrolleitungen.

Die Kontrolleitungen werden in Punkt 1.2 besprochen.
Die insgesamt 16 Adressenleitungen werden zu dem Adressenbus zusammengefaßt. Er ist 16 Bit breit; das heißt, es können zwei Bytes parallel über ihn geschickt werden. Mit diesen 16 Bit kann der Z80 Adressen im Bereich von 0 bis 65535 anwählen.
Wenn nun der Z80 arbeitet, benötigt er ständig Daten aus Speicherzellen und muß laufend Daten in Speicherzellen schicken.
Wenn etwas aus einer Speicherzelle gelesen werden soll, muß zuerst die Adresse dieser Speicherzelle über den Adressbus angewählt werden, dann schickt der Speicherbaustein, in dem die Speicherzelle liegt, dessen Inhalt über den Datenbus zurück zur Z80. Falls etwas in einer Speicherzelle geschrieben werden soll, wird auch zuerst die Adresse über den Adressbus geschickt, dann das, was hineingeschrieben werden soll, über den Datenbus gesendet.
Der Datenbus besteht aus 8 Datenleitungen und umfaßt ebenfalls 8 Bits bzw. 2 Bytes, da eine Speicherzelle nur diese Speicherkapazität hat.

1.2 Die Kontrolleinheit

Die Kontrolleinheit des Z80 ist dafür verantwortlich, daß im Z80 alles wie vorgesehen abläuft. Die Kontrolleinheit sorgt für die richtige Adressierung der Speicherzellen, damit der Z80 aus ihnen Daten lesen bzw. in sie hineinschreiben kann.
Allgemein werden im Z80 durch die Kontrolleinheit Kontrollsignale erzeugt, wobei manche nur für den Z80 intern gebraucht und andere über die externen Kontrolleitungen geschickt werden.
Über die Kontrolleitungen wird z.B. durch Signale angezeigt, ob aus Speicherzellen gelesen oder in sie hineingeschrieben werden soll.
Man sollte beachten, daß die Kontrolleinheit nicht entscheidet, was sie zu tun hat, sondern daß sie nur ausführt. Der Z80 hat sich nach dem Maschinencodeprogramm zu richten, also auch die Kontrolleinheit.

1.3 Die Recheneinheit

Sie bearbeitet arithmetische und logische Funktionen. Die einzigen Vorgänge, die die Recheneinheit ausführen kann, sind Addition und Subtraktion. Multiplikation und Division sind viel zu schwierig für die Recheneinheit; außerdem stellen sie eine Wiederholung der Addition und Subtraktion dar.

1.4 Das Befehlsregister

Wenn der Z80 ein Programm abarbeitet, wird jeder Befehl, der gerade bearbeitet wird, ins Befehlsregister geladen.

1.5 Der Programmzähler

Der Programmzähler ist ein Doppelregister im Z80. Er zeigt auf die Speicherzelle, in der der Befehl steht, der gerade abgearbeitet wird oder der als nächster in das Befehlsregister geladen werden soll.
Wenn nun ein Befehl geladen wird, hat die Kontrolleinheit dafür zu sorgen, daß der Programmzähler erhöht wird und damit auf den nächsten Befehl zeigt.

1.6 Die Hauptegister

Es gibt im Z80 noch 24 andere Register, die vom Programmierer benutzt werden können. Die Namen der 24 Register sind nicht nach logischen Gesichtspunkten gewählt worden. Manche stammen noch von Vorgängern des Z80 ab, die noch nicht so kompliziert waren. Alle 24 Register sind Ein-Byte-Register, die (wie Abb. 3 zeigt) zu Doppelregistern kombiniert werden können.

1.6.1 Das A-Register

Dies ist das wichtigste Register des Z80 und wird auch Akkumulator genannt. Diese Bezeichnung stammt noch aus der Zeit, als es in Mikroprozessoren nur ein einziges Register gab, das ein Ergebnis speichern konnte. Im Z80 wird das A-Register hauptsächlich für arithmetische und logische Operationen benutzt, doch auch viele andere Funktionen (wie z.B. Vergleichsfunktionen) können nur mit dem A-Register durchgeführt werden.

1.6.2 Das F-Register

Das F-Register wird auch FLAG-Register genannt. Die Anwendung der Flags wird in einem späteren Kapitel behandelt
Ein Flag ist ein Bit, das in Abhängigkeit von einer Operation entweder 0 (ungesetzt) oder 1 (gesetzt) enthält. Das F-Register enthält 8 Bits; jedoch kann vom Programmierer nur auf 4 Flags zurückgegriffen werden. Dies sind

- das Zero Flag
- das Sign Flag
- das Parity/Overflow Flag.

Die anderen Flags werden nur von der Kontrolleinheit benutzt.

1.6.3 Das HL-Registerpaar

Dieser Name stammt auch von früheren Prozessoren.
Als man zwei Register zu einem Paar zusammenfaßte, benannte man die Register nach ihrer Priorität. Das H-Register enthält demnach das 'high' (höherwertige) Byte, das L-Register das 'low' (niederwertige) Byte. Die meisten Operationen, die auf Adressen zugreifen, können nur mit dem HL-Registerpaar durchgeführt werden.

1.6.4 Die Registerpaare BC und DE

Diese Registerpaare sind jüngeren Datums und sind zusätzliche Registerpaare ohne spezielle Funktionen.

1.6.5 Das IX-Registerpaar

Dieses Register wird für indexadressierte Speicherzugriffe benutzt. Im Register IX steht ein Grundwert, dessen Entfernung zur eigentlichen Adresse, die angesprochen werden soll, durch ein Zweierkomplement-Datenbyte angegeben wird.

1.6.6 Das IY-Registerpaar

Die Funktionsweise ist die gleiche wie beim IX-Registerpaar.

1.6.7 Der Stackpointer (Stapelzeiger)

Der Stapelzeiger ist ein Registerpaar, dessen Inhalt auf eine Stelle in einem Speicherteil zeigt, der Stack (Stapel) genannt wird. Dieser dient zum zeitweiligen Festhalten von Adressen, Rücksprungadressen oder Daten.
Der Stapel des Z80 wird von oben nach unten im Speicher angelegt, das heißt, der Z80 baut seinen Stapel an der für ihn höchstmöglich verfügbaren Adresse auf. Dabei werden neue Elemente, die für den Stapel bestimmt sind, unter die anderen gesetzt. Das Element, das zuletzt auf den Stapel gebracht wurde, wird auch zuerst wieder ausgelesen.
Der Stapelzeiger zeigt immer auf die Adresse der zuletzt auf den Stapel gebrachten Daten; er wird um zwei erhöht, wenn ein Datenelement ausgelesen wird, um zwei erniedrigt, wenn Neues auf den Stapel gebracht wird.
Falls Daten für kurze Zeit festgehalten werden sollen, empfiehlt es sich eher, sie auf den Stapel zu bringen, als sie in eine Speicherzelle zu schreiben, weil diese Methode wesentlich schneller ist.

1.6.8 Das R-Register

Dieses Register ist das Refresh-Register. Es besteht nur aus einem Zähler, der jedesmal um Eins erhöht wird, wenn Befehle oder Daten aus einer Speicherstelle gelesen werden.
Der Wert im R-Register läuft also immer von 0 bis 255. Normalerweise dient dieses Register dazu, anzuzeigen, welche Speicherstellen aufgefrischt werden müssen. Beim ZX81 wird das R-Register benutzt, um zu zählen, wie viele Zeichen benötigt werden, um eine Zeile auf dem Bildschirm darzustellen.

1.6.9 Das I-Register

Dies ist das Interrupt-Register
Wenn der Z80 ein Programm abarbeitet, kann es unterbrochen werden. Solche Interrupts werden meist von externen Geräten erzeugt, z.B. von einem Drucker. Beim ZX81 werden diese Interrupts nur zur Generierung des Bildes benutzt.
Wenn nun ein Gerät den Z80 unterbrechen will, sendet es ein Signal über die entsprechende Kontrolleitung. Wenn der Z80 das Signal empfängt, stoppt er sofort die Programmausführung und springt zur Adresse, deren höherwertiges Byte das I-Register darstellt.
Beim ZX81 springt der Z80 im Falle eines Interrupts entweder nach 0038h oder nach 0066h.

2. Grundlagen der Maschinencodebefehle

Maschinencodebefehle werden normalerweise in Assemblersprache geschrieben, das heißt, jeder Befehl wird durch ein Mnemonikkürzel für den Programmierer lesbar gemacht. Da der ZX81 als Computersprache BASIC kennt, ist es sinnvoll, Maschinencodebefehle anhand von BASIC-Instruktionen zu erklären. Trotzdem sind BASIC- und Maschinencodeprogramme von der Struktur her ziemlich verschieden.
Ein BASIC-Programm besteht aus einer Anzahl von BASIC-Zeilen. Jede Zeile besteht aus einer Zeilennummer und einem Befehl. Die Zeilennummern im BASIC-Programm zeigen die Reihenfolge der abzuarbeitenden Befehl an; im Maschinencode gibt es Adressen statt Zeilennummern. Die Adressen werden in aufsteigender Reihenfolge abgearbeitet. BASIC verfügt über 59 Befehle und Funktionen, in Maschinencode existieren über 500 Befehle. Jeder dieser Befehle kann 1 - 4 Bytes lang sein, wobei die Anweisung selbst nur 1 - 2 Bytes umfassen kann. Die restlichen Bytes können Daten sein, müssen aber nicht unbedingt vorkommen.

Es gibt keinen Z80-Befehl, der mehr als 4 Bytes umfaßt.

Jeder Befehl kann mnemonisch ausgedrückt werden, das heißt, der Programmierer kann durch das Kürzel sehen, was der Befehl veranlaßt. So bedeutet z.B. LD A,(02B4) hexadezimal 7E B4 02, was nur mit viel Übung als Ladebefehl erkannt werden kann. Die Anweisung LD A,(02B4) veranlaßt den Z80, den Inhalt der Adresse 02B4 (daher die beiden Klammern) in das Register A zu laden (LD).
Wir gehen davon aus, daß die Leser dieses Buches BASIC beherrschen. Im Nachfolgenden wird versucht, jeden Maschinencodebefehl anhand eines oder mehrerer BASIC-Befehle zu erklären. Dabei läßt sich sofort die Funktionsweise erkennen.

2.1 Rechnen mit binären Zahlen

Wie bereits erklärt, ist der Z80 in der Lage, gleichzeitig 8 oder 16 Bits zu verarbeiten. Jedes Byte kann Werte von 0000 0000 bis 1111 1111 bin annehmen (0 - 255 dez), wobei es keine Möglichkeit gibt, negative Zahlen von 0 bis -255 durch Bits oder ein Byte darzustellen. Es ist sehr wichtig zu wissen, daß alle Register und Speicherzellen im Z80 nur positive Werte annehmen können. Um nun aber auch negative Werte darstellen zu können, wurde die Zweierkomplementierung angewandt. Das Prinzip ist relativ einfach:
Bit 7 fungiert dabei als Vorzeichen eines Bytes. Ist es 0 (zurückgesetzt oder ungesetzt), so ist die Zahl positiv; ist es 1 (gesetzt), so ist die Zahl negativ. Auf diese Weise kann man jede Zahl zwischen -128 und 127 darstellen.
Die binären Werte von 0 bis 127 werden weiterhin normal dargestellt:

```
0000 0000 bin = 0 dez
0000 0001 bin = 1 dez
0000 0010 bin = 2 dez
usw.
```

Die negativen Zahlen sehen binär dann so aus:

```
1111 1111 bin = -1 dez
 1111 1110 bin = -2 dez
 1111 1101 bin = -3 dez
usw.
```

Es gibt einen einfachen Weg, das Zweierkomplement einer Zahl zu berechnen: man nimmt die Zahl (z.B. -13) und formt sie in die binäre positive Darstellung um:

13 dez = 0000 1101 bin.

Nun ändert man jedes Bit in sein Gegenteil, also wird aus 0 eine 1, und umgekehrt.

0000 1101 ergibt dann 1111 0010.

Nun wird noch 1 dazuaddiert:

1111 0010 + 0000 0001 = 1111 0011.

Zum Schluß formt man wieder in die dezimale Darstellung um:

1111 0011 bin = 243 dez.

Nun ist es natürlich recht schwierig, immer mit binären Zahlen zu arbeiten; auch Dezimalzahlen sind nicht immer vorteilhaft. Da ein Byte genau 256 verschiedene Werte annehmen kann, wäre es günstig, alle 256 möglichen Werte mit zwei Zeichen darstellen zu können. Hierzu eignet sich das Hexadezimalsystem (Basis 16) besonders gut, denn 16 x 16 = 256.
Man kann jedes Byte nun in zwei Teile mit je 4 Bits aufteilen. Diese 4 Bits werden 'nibble' genannt. Jedes 'nibble' kann man jetzt mit den Ziffern 0-9 und A-F (= 10-15) bezeichnen:

0000 bin = 0 hex = 0 dez
0001 bin = 1 hex = 1 dez
0010 bin = 2 hex = 2 dez
0011 bin = 3 hex = 3 bin
0100 bin = 4 hex = 4 dez
0101 bin = 5 hex = 5 dez
0110 bin = 6 hex = 6 dez
0111 bin = 7 hex = 7 dez
1000 bin = 8 hex = 8 dez
1001 bin = 9 hex = 9 dez
1010 bin = A hex = 10 bin
1011 bin = B hex = 11 dez
1100 bin = C hex = 12 dez
1101 bin = D hex = 13 dez
1110 bin = E hex = 14 dez
1111 bin = F hex = 15 dez.

Nun sieht z.B. 1010 1010 bin hexadezimal so aus: AA hex. Oder 0000 0001 bin = 01 hex.

Ebenso einfach läßt sich jetzt auch eine Zwei-Byte-Zahl darstellen; sie wird einfach in vier nibbles aufgeteilt. Beispiel:

0111 1001 0100 1100 bin = 794C hex.

Das Umrechnen von hexadezimal zu dezimal ist ebenso recht einfach:

Gegeben sei eine Hexadezimalzahl, z.B. WXYZ.

$$\text{WXYZ hex} = W \times 16^3 + X \times 16^2 + Y \times 16^1 + Z \times 16^0$$
$$= W \times 1094 + X \times 256 + Y \times 16 + Z \text{ dez}$$

Bei Ein-Byte-Zahlen der Form XY:

XY hex = X x 16 + Y dez.

3. Die Z80-Maschinencodebefehle

In diesem Kapitel werden alle Maschinencodebefehle erklärt. Es wird versucht, diese Befehle durch BASIC-Befehle deutlich zu machen, um die Funktionsweise verständlich werden zu lassen. Außerdem wird auch anhand von kurzen Programmen in Maschinencode mancher Befehl deutlich gemacht.
Um nun diese Programme laufen lassen zu können, geben Sie bitte folgendes Programm ein:

```
1       REM 01234567890123456789012345678 9
10      FOR A=16514 TO 16543
20      SCROLL
30      PRINT A,
40      INPUT C
50      PRINT C
60      POKE A,C
70      NEXT A
```

Die Maschinencodeprogramme werden im REM-Statement in Zeile 1 abgelegt, wenn sie über die Schleife eingegeben werden.

3.1 Die einzelnen Flags

3.1.1 Das Zero (Null-) Flag

Dieses Flag ist Bit 6 des F-Registers. Es wird immer dann gesetzt, wenn bei einer Operation der Wert Null herauskommt, sonst wird er auf Null gesetzt. Alle Befehle wie ADC, ADD, AND, CP, DEC, INC, OR, SUB, SBC und XOR beeinflussen das Zero Flag, ebenso SET, RES und BIT, auch Bitverschiebebefehle, ausgenommen Ladebefehle.

3.1.2 Das Sign (Vorzeichen-) Flag

Dies ist Bit 7 des F-Registers. Es zeigt an, ob eine Zahl negativ oder positiv ist (s. unter Zweierkomplement). Das Sign Flag entspricht damit Bit 7 eines 1-Byte-Wertes bzw. Bit 15 eines 2-Byte-Wertes. Die Befehle, die das Sign Flag beeinflussen, sind die gleichen wie beim Zero Flag.

3.1.3 Das Parity/Overflow Flag

Dies ist Bit 2 des F-Registers. Es testet entsprechend des vorausgegangenen Befehls, ob eine gerade oder ungerade Anzahl von gesetzten Bits in dem entsprechenden Register vorliegt (Parity), oder es wird bei Befehlen wie ADD, ADC, SBC und SUB benutzt, um zu testen, ob bei einer Zweierkomplementzahl bei Überschreitung von 0 auch ein richtiges Ergebnis herauskommt.
Angenommen, man addiere 100 + 100, so kommt 200 heraus; dies geschieht aber nur bei absoluten Werten. Das Zweierkomplement von 200 ist aber nie 200, sondern, wie Sie sich sicher erinnern werden, eine negative Zahl, nämlich -54. Falls ein Überlauf dieser Art entsteht, wird das P/O Flag gesetzt.
Bei den Befehlen AND, OR und XOR wird getestet, ob das entsprechende Register eine gerade oder ungerade Anzahl von gesetzten Bits enthält. Das Flag wird gesetzt, wenn die Anzahl gerade ist.
Das P/O Flag wird außerdem gesetzt, wenn bei einem INC der Wert 128 herauskommt, bei DEC der Wert 127.

3.1.4 Das Carry Flag

Das Carry Flag ist Bit 0 des F-Registers.
Es wird gesetzt, wenn bei einer Addition ein zu großes Ergebnis herauskommt, oder wenn bei einer Subtraktion ein negatives Ergebnis herauskommt. Das Carry Flag wird von folgenden Befehlen beeinflußt:

ADC, ADD, AND, CP, OR, SBC, SUB und XOR.

Ladebefehle beeinflussen das Carry Flag auch hier nicht (egal, ob gesetzt oder nicht gesetzt).
Es gibt zwei Befehle, mit denen man das Carry Flag verändern kann:

SCF (Setze Carry Flag),
CCF (Komplementiere Carry Flag), d.h. wenn das Flag 0 ist, wird es zu 1, und umgekehrt.

3.2.1 NOP

Der No-Operation-Befehl veranlaßt den Z80, 1 Mikrosekunde (µsec) lang nichts zu tun.

3.2.2 RET

Der Return-Befehl veranlaßt den Z80, von einem Unterprogramm zu seinem Aufruf zurückzuspringen, der mit CALL (s. dort) aufgerufen wurde. CALL und RETURN entsprechen den BASIC-Befehlen GOSUB und RETURN.
RET springt auch, wenn die im Programm vorkommenden CALL-Aufrufe abgearbeitet wurden, an die Stelle ins BASIC zurück, an der es mit USR aufgerufen wurde.

Beispiel:

```
16514   NOP         0
16515   RET         201
```

Sie brauchen nur das in Kapitel 3 aufgeführte Programm einzugeben, zu starten und dann die ganz rechts stehenden Daten einzugeben. Zum Schluß wird STOP (NEWLINE) eingegeben, damit Sie aus der Schleife herauskommen - und das Ganze wird mit RAND USR 16514 gestartet. Das Maschinencodeprogramm führt einen Leerbefehl aus (NOP), dann springt es wieder ins BASIC zurück (RET).

3.2.3 LD

3.2.3.1 Laden eines Registers mit einer Konstanten

Jedes Register kann mit einer Konstanten geladen werden. Ein-Byte-Konstanten werden im folgenden mit 'dd' bezeichnet, Zwei-Byte-Konstanten mit 'dddd'. Eine Variable, in die jedes Ein-Byte-Register eingesetzt werden kann, ist **r**. Die Variable für die Registerpaare BC, DE, HL, IX, IY und SP ist **rr**. - Die Befehle sind:

```
LD A,dd                 3E dd hex
LD B,dd                 06 dd hex
LD C,dd                 0E dd hex
LD D,dd                 16 dd hex
LD E,dd                 1E dd hex
LD H,dd                 26 dd hex
LD L,dd                 2E dd hex
```

Beispiel:

```
16514   LD B,00         6  0
16516   LD C,10         14 16
16518   RET             201
```

Dieses Programm sollte man mit PRINT USR 16514 starten; danach erscheint links oben auf dem Bildschirm eine 16.

Wie Sie vielleicht schon gemerkt haben, kann man ein Maschinencodeprogramm nur mit Befehlen aufrufen, die ein numerisches Argument verlangen. Dieses numerische Argument stellt USR xxxxx dar. Nun kommt aber noch etwas als Ergebnis von PRINT USR xxxxx heraus, und **das** ist der Inhalt des **BC-Registerpaares.** (RAND USR xxxxx arbeitet das angesprochene Maschinencodeprogramm ohne den Ausdruck des Inhaltes des BC-Registers ab)
Ruft man obiges Maschinencodeprogramm z.B. mit LET A=USR 16514 auf, so hat A nun den Wert 16 (die Variable A, nicht das Register!).

Nun zum eigentlichen Maschinencodeprogramm:
Als erstes wird das BC-Doppelregister mit 16 geladen. B ist das höherwertige Byte; es wird mit 0 geladen, da der Wert nicht größer als 255 ist. Nun wird C mit 16 (10 hex) geladen, anschließend wird ins BASIC-Programm zurückgesprungen. Man kann obiges Programm durch folgende BASIC-Zeilen erklären:

```
LET BC = 16
PRINT BC
```

3.2.3.2 Laden eines Registers in ein anderes

Jedes der Register A, B, C, D, E, H und L kann in ein anderes dieser Register geladen werden. Das R- und I-Register kann nur in den Akkumulator und umgekehrt geladen werden.

	LD A,	LD B,	LD C,	LD D,	LD E,	LD H,	LD L,	LD I,	LD R,
A	7F	47	4F	57	5F	67	6F	ED 47	ED 4F
B	78	40	48	50	58	60	68	-	-
C	79	41	49	51	59	61	69	-	-
D	7A	42	4A	52	5A	62	6A	-	-
E	7B	43	4B	53	5B	63	6B	-	-
H	7C	44	4C	54	5C	64	6C	-	-
L	7D	45	4D	55	5D	65	6D	-	-
I	ED 57	-	-	-	-	-	-	-	-
R	ED 5F	-	-	-	-	-	-	-	-

Nun dazu ein Beispiel:

```
16514   LD C,01         14  1
16516   LD B,C          65
16517   RET             201
```

Ins C-Register wird 1 geladen, danach wird die 1 ins B-Register dupliziert. Wenn man das Programm mit PRINT USR 16514 startet, muß 257 gedruckt werden (B x 256 + C).

3.2.3.3 Laden eines Doppelregisters mit einer Konstanten

```
LD BC,dddd          01 dd dd
LD DE,dddd          11 dd dd
LD HL,dddd          21 dd dd
LD SP,dddd          31 dd dd
LD IX,dddd          DD 21 dd dd
LD IY,dddd          FD 21 dd dd
```

Doppelregisterladebefehle laden die nächsten zwei Datenbytes ins Doppelregister, und zwar das erste Byte in das niederwertige Register (C,E,L,P,X,Y), das zweite Byte ins höherwertige Register (B,D,H,I,S).

Im nachfolgenden Beispiel wird das Programm aus Kapitel 3.2.3.1 vereinfacht: um 16 ins BC-Register zu laden, benutzt man nun folgendes Programm:

```
16514   LD BC,0010      1   16   0
16517   RET             201
```

Hier wird 16 dez direkt ins Doppelregister geladen, was 1 Byte spart. Bitte denken Sie immer daran, daß zuerst das niederwertige Byte codiert werden muß und gelesen wird!

3.2.3.4 Laden eines Doppelregisters in ein anderes

Dafür gibt es nur drei Befehle, die sich alle auf das SP-Register beziehen. Alle anderen Registerumladungen können nämlich durch Einzelregisterumladungen ersetzt werden (z.B. LD BC,HL = LD B,H und LD C,L). - Die drei Befehle sind:

```
LD SP,HL            F9
LD SP,IX            DD F9
LD SP,IY            FD F9
```

Diese drei Befehle sprechen wohl für sich selbst und brauchen daher nicht näher erklärt zu werden.

3.2.3.5 Laden eines Registers oder Doppelregisters mit dem Inhalt einer Speicheradresse

Der Z80 besitzt viele Befehle, die Daten aus einer Speicherstelle holen und in ein Register bringen. Jeder dieser Befehle muß also zwei Angaben enthalten:

- die Adresse, von der die Daten geholt werden sollen
- das Register, in das die Daten gebracht werden sollen.

Die Klammern um die Adresse zeigen an, daß als Wert nicht die Adresse selbst, sondern deren Inhalt geladen wird.

3.2.3.5.1 Absolute Adressierung

Bei folgenden Befehlen steht die Adresse fest, aus der die Daten geholt werden sollen.

```
LD A,(addr)          3A addr
LD HL,(addr)         2A addr     oder     ED 6B addr
LD BC,(addr)         ED 4B addr
LD DE,(addr)         ED 5B addr
LD IX,(addr)         DD 2A addr
LD IY,(addr)         FD 2A addr
LD SP,(addr)         ED 7B addr
```

Auch hier zeigt sich, daß nur ein einzelnes Byte in den Akkumulator geladen werden kann. Bei den anderen Befehlen wird in das niederwertige Byte der Inhalt von addr geladen, in das höherwertige der Inhalt von addr+1. Selbstverständlich wird auch bei der Adressencodierung das niederwertige Byte vorangesetzt. - Um nun so einen Doppelregisterladebefehl zu verdeutlichen, ein Beispiel in BASIC:

Der Maschinencodebefehl LD BC,(16396) lädt den Inhalt der Adressen 16396 und 16397 (D-File) ins BC-Doppelregister. Nun das gleiche mit BASIC:

```
LET C=PEEK (16396)
LET B=PEEK (16396+1)
```

Und noch ein Maschinencode-Beispielprogramm:

```
16514   LD BC,(16388)   237  75  4  64
16518   RET             201
```

Dieses Programm wird auch wieder mit PRINT USR 16514 gestartet. Das Ergebnis zeigt, wieviel RAM am ZX steckt, wenn man 16383 abzieht.

3.2.3.5.2 Indirekte Adressierung

Bei der indirekten Adressierung wird der Inhalt aus einer Speicherstelle geholt, deren Adresse in einem Doppelregister festgehalten ist. Dieses Doppelregister kann mit beliebigen Adressen geladen werden, so daß man mit diesen Befehlen den ganzen Speicher adressieren kann.

```
LD A,(HL)            7E
LD A,(BC)            0A
LD A,(DE)            1A
LD B,(HL)            46
LD C,(HL)            4E
LD D,(HL)            56
LD E,(HL)            5E
LD H,(HL)            66
LD L,(HL)            6E
```

Hierzu ein Beispiel:

```
16514   irgendein Wert zwischen 0 und 255
16515   LD HL,16514     33  130  64
16518   LD B,0          6  0
16520   LD C,(HL)       78
16521   RET             201
```

Dieses Programm mit PRINT USR **16515** starten!

Es wird der Wert gedruckt, der in 16514 steht. Zuerst wird in HL die Adresse geladen, aus der Daten geholt werden sollen, dann wird der Inhalt der Adresse in HL in C geladen.

3.2.3.5.3 Index-Adressierung

Die Indexadressierung wird dazu benutzt, um Daten z.B. aus Tabellen oder Listen auszulesen. Mit den Indexregistern IX oder IY kann man auf Speicherblöcke mit der Größe 256 Bytes zeigen. Die Basisadresse steht im Indexregister, dazu kommt noch ein Zweierkomplement-Datenbyte, welches je nach Wert vom Indexregister abgezogen oder dazuaddiert wird. Wenn das Datenbyte 0 ist, so zeigt das Indexregister direkt auf die Speicherstelle.

```
        IX + 7F
        IX + 7E
        IX + 7D
        IX + 7C
        ...
        ...
        IX + 02
        IX + 01
IX ---> IX + 00
        IX + FF
        IX + FE
        ...
        ...
        IX + 81
        IX + 80
```

Hier die Auflistung der Befehle:

LD A,(IX+dd)	DD 73 dd	LD A,(IY+dd)	FD 73 dd
LD B,(IX+dd)	DD 46 dd	...	
LD C,(IX+dd)	DD 4E dd	...	
LD D,(IX+dd)	DD 56 dd		
LD E,(IX+dd)	DD 5E dd		
LD H,(IX+dd)	DD 66 dd		
LD L,(IX+dd)	DD 6E dd		

Um Befehle vom IX- aufs IY-Register umzucodieren, braucht man nur DD in FD zu ändern.

Es ist schwer, beim ZX81 den Inhalt von IX und IY zu ändern, weil dadurch der Rechner meistens abstürzt. Nun zeigt das IY-Register auf den Anfang des RAMs (16384), und da das Byte 16514 (also das erste Zeichen nach dem REM) mehr als 127 Bytes weit entfernt ist, nehmen wir in diesem Beispiel die Adresse 16508 und POKEn dort einen Wert zwischen 0 und 255 ein. - Das Maschinenprogramm zum Auslesen dieses Bytes sieht dann so aus:

16514	LD C,(IY+7C)	253 78 124
16517	LD B,00	6 0
16519	RET	201

Dieses Programm wird wieder mit PRINT USR 16514 gestartet; herauskommen sollte der Wert, der in 16444 steht.

3.2.3.6 Laden einer Speicheradresse mit dem Inhalt eines Registers

Diese Befehle sind sozusagen die 'Umkehrfunktionen' zu den Befehlen in Kapitel 3.2.3.5.

3.2.3.6.1 Absolute Adressierung

```
LD (addr),A          32 addr
LD (addr),HL         22 addr   oder   ED 63 addr
LD (addr),BC         ED 43 addr
LD (addr),DE         ED 53 addr
LD (addr),IX         DD 22 addr
LD (addr),IY         FD 22 addr
LD (addr),SP         ED 73 addr
```

Es gibt keinen Befehl, mit dem man einen konstanten Wert direkt in eine konstante Adresse laden kann. - Beispiel:

```
16515   LD A,3F          62  63
16517   LD (4082),A      50  130  64
16520   RET              201
```

Nachdem das Programm mit RAND USR 16515 gestartet wurde, muß das Zeichen nach der REM-Anweisung ein **Z** sein.
In A wurde der Code von Z (63 dez) geladen, dann wurde A in die Speicherstelle 16514 geladen.

3.2.3.6.2 Indirekte Adressierung

```
LD (HL),A            77
LD (BC),A            02
LD (DE),A            12
LD (HL),B            70
LD (HL),C            71
LD (HL),D            72
LD (HL),E            73
LD (HL),H            74
LD (HL),L            75
```

Zu diesen Befehlen bedarf es wohl keiner besonderen Erklärung.

3.2.3.6.3 Index-Adressierung

```
LD (IX+dd),A         DD 77 dd
LD (IX+dd),B         DD 70 dd
LD (IX+dd),C         DD 71 dd
LD (IX+dd),D         DD 72 dd
LD (IX+dd),E         DD 73 dd
LD (IX+dd),H         DD 74 dd
LD (IX+dd),L         DD 75 dd
LD (IX+d1),d2        DD 36 d1 d2
```

Wie auch bei den vorhergehenden Index-Befehlen, muß hier beim Gebrauch des IY-Registers DD in FD geändert werden.

3.2.4 Registeraustausch

Es gibt drei Befehle, die bestimmte Register direkt gegeneinander austauschen.

```
EX DE,HL             EB
EXX                  D9
EX AF,A'F'           08
```

Der Befehl EX DE,HL spricht für sich; DE wird in HL geladen und umgekehrt.

Der EXX-Befehl tauscht den Hautpregistersatz gegen den Schwesterregistersatz aus, mit Ausnahme A und F, also B, C, D, E, H und L gegen B', C', D', E', H' und L'. Der Befehl EX AF,A'F' dagegen tauscht nur das A gegen das A'- und das F- gegen das F'-Register aus.
Hierzu ein Beispiel:

```
16514   LD HL,0010      33  16  0
16517   EX DE,HL        235
16518   LD B,D          66
16519   LD C,E          75
16520   RET             201
```

Dieses Programm wird wieder mit PRINT USR 16514 gestartet. Wieder soll 16 erscheinen. Zuerst wird HL mit 16 geladen, dann wird DE gegen HL ausgetauscht, d.h. DE enthält jetzt 16, HL den früheren Wert von DE. Nun wird B mit D (also 0) geladen und C mit E (16).

3.2.4.2 Austausch eines Doppelregisters gegen den Inhalt einer Speicherstelle

```
EX (SP),HL          E3
EX (SP),IX          DD E3
EX (SP),IY          FD E3
```

Mit diesen Befehlen werden die beiden letzten Bytes des Stapels gegen den Inhalt eines Doppelregisters ausgetauscht.

3.2.5 Rechenoperationen

3.2.5.1 Inkrementieren

Wenn ein INC-Befehl abgearbeitet wird, so wird das entsprechende Register oder Doppelregister um 1 erhöht. Das Carry Flag wird durch INC nicht verändert.

```
INC A               3C
INC B               04
INC C               0C
INC D               14
INC E               1C
INC H               24
INC L               2C
INC (HL)            34
INC (IX+dd)         DD 34 dd
INC (IY+dd)         FD 34 dd
INC BC              03
INC DE              13
INC HL              23
INC SP              33
INC IX              DD 23
INC IY              FD 23
```

Hierzu ein Beispiel:

```
16514   LD BC,0010      1   16   0
16517   INC BC          3
16518   INC BC          3
16519   RET             201
```

Nach PRINT USR 16514 muß 18 erscheinen. Zuerst wurde BC mit 16 dez geladen und durch die zwei INC BC-Befehle um zwei erhöht, also auf 18.

3.2.5.2 Dekrementieren

Beim DEC-Befehl wird das entsprechende Register oder Doppelregister um 1 erniedrigt.

```
DEC A               3D
DEC B               05
DEC C               0D
DEC D               15
DEC E               1D
DEC H               25
DEC L               2D
DEC (HL)            35
DEC (IX+dd)         DD 35 dd
DEC (IY+dd)         FD 35 dd
DEC BC              0B
DEC DE              1B
DEC HL              2B
DEC SP              3B
DEC IX              DD 2B
DEC IY              FD 2B
```

Auch hierzu ein Beispiel:

```
16514   LD BC,0010      1   16   0
16517   DEC BC          11
16518   DEC BC          11
16519   DEC BC          11
16520   RET             201
```

Hier wird BC erst mit 16 dez geladen und durch dreimaliges DEC BC um drei erniedrigt. Nach PRINT USR 16514 müßte also 13 herauskommen. Wie beim INC-Befehlssatz haben die DEC-Befehle keinen Einfluß auf das Carry Flag.

3.2.5.3 Einfache Addition

```
ADD A,dd            C6 dd
ADD A,A             87
ADD A,B             80
ADD A,C             81
ADD A,D             82
ADD A,E             83
ADD A,H             84
ADD A,L             85
ADD A,(HL)          86
ADD A,(IX+dd)       DD 86 dd
```

```
ADD HL,BC          09
ADD HL,DE          19
ADD HL,HL          29
ADD HL,SP          39
ADD IX,IX          DD 29
ADD IX,BC          DD 09
ADD IX,DE          DD 19
ADD IX,SP          DD 39
```

Hierbei braucht eigentlich nicht mehr erwähnt zu werden, daß DD gegen FD ausgewechselt werden muß, wenn IY statt IX verwendet werden soll. - Hierzu zwei Beispiele:

1. Ein-Byte-Addition

```
16514   LD B,00        6   0
16516   LD A,10        62  16
16518   LD C,30        14  48
16520   ADD A,C        129
16521   LD C,A         79
16522   RET            201
```

Zuerst wird das B-Register mit 0 geladen, damit kein Wert größer als 255 ausgegeben wird, was hiermit zum letzten Mal erklärt wird. Dann wird A mit 16 dez geladen, und C mit 48 dez. A und C werden addiert, das Ergebnis kommt nach A. Da wir es aber in C brauchen, wird es einfach nach C geladen.
Auch dieses Programm wird mit PRINT USR 16514 gestartet.

2. Zwei-Byte-Addition:

```
16514   LD BC,4000     1   0   64
16517   LD HL,2000     33  0   32
16520   ADD HL,BC      9
16521   LD B,H         68
16522   LD C,L         77
16523   RET            201
```

Hier wird 4000 hex und 2000 hex addiert. Nach PRINT USR 16514 müßte 6000 hex (also 24576 dez) gedruckt werden. Bei diesem Programm wird A praktisch durch HL ersetzt, und C durch BC.

3.2.5.4 Einfache Subtraktion

Die SUB-Befehle beziehen sich **alle** auf den Akkumulator, so daß er im Mnemonik nicht mit angegeben wird.

```
SUB dd             D6 dd
SUB A              97
SUB B              90
SUB C              91
SUB D              92
SUB E              93
SUB H              94
SUB L              95
SUB (HL)           96
SUB (IX+dd)        DD 96 dd
SUB (IY+dd)        FD 96 dd
```

Auch hierzu ein Beispiel:

```
16514   LD B,00         6   0
16516   LD A,10         62  16
16518   SUB 09          214 9
16520   LD C,A          79
16521   RET             201
```

Nach PRINT USR 16514 muß 7 gedruckt werden. A wird mit 16 dez geladen, davon wird 9 dez subtrahiert. In A bleibt 7 zurück, was dann in C geladen wird.

3.2.5.5 Addition mit Carry

Die Addition mit dem Carry Flag verhält sich bei ungesetztem Carry Flag genauso wie die normalen ADD-Befehle. Ist das Carry Flag gesetzt, wird das Ergebnis einmal inkrementiert. - Beispiel:

```
Carry ungesetzt     20 dez ADC 10 dez = 20 + 10 = 30
Carry gesetzt       20 dez ADC 10 dez = 20 + 10 + 1 = 31
```

Die Befehlsauflistung:

```
ADC A,dd            CE dd
ADC A,A             8F
ADC A,B             88
ADC A,C             89
ADC A,D             8A
ADC A,E             8B
ADC A,H             8C
ADC A,L             8D
ADC A,(HL)          8E
ADC A,(IX+dd)       DD 8E dd
ADC A,(IY+dd)       FD 8E dd
ADC HL,HL           ED 6A
ADC HL,BC           ED 4A
ADC HL,DE           ED 5A
ADC HL,SP           ED 7A
```

3.2.5.6 Subtraktion mit Carry

Die Subtraktionen mit dem Carry Flag verhalten sich bei ungesetztem Carry Flag genauso wie die normalen SUB-Befehle. Bei gesetztem Carry Flag wird das Ergebnis einmal dekrementiert.

```
SBC A,dd            DE dd
SBC A,A             9F
SBC A,B             98
SBC A,C             99
SBC A,D             9A
SBC A,E             9B
SBC A,H             9C
SBC A,L             9D
SBC A,(HL)          9E
SBC A,(IX+dd)       DD 9E dd
SBC A,(IY+dd)       FD 9E dd
SBC HL,HL           ED 62
SBC HL,DE           ED 52
SBC HL,SP           ED 72
```

Die ADC- und SBC-Befehle werden hauptsächlich zur Fließkommaberechnung verwendet.

3.2.6 Vergleichsoperatoren

Die Vergleichsoperatoren erlauben dem Programmierer, den Inhalt des Akkumulators mit anderen Daten zu vergleichen. Dies geschieht, indem eine Subtraktion simuliert wird und nach dem Ergebnis die Flags gesetzt werden.
So wird z.B. das Carry Flag gesetzt, wenn der Akkumulator kleiner als das Vergleichsbyte ist; es wird das Zero Flag gesetzt, wenn der Akkumulator gleich dem Vergleichsbyte ist.
Zu diesem Zeitpunkt können noch keine Beispiele zu Operationen, die nur den Sinn der Flagbeeinflussung haben, gebracht werden, da die Befehle, die die Flags abfragen, noch nicht erklärt sind. Hier die Befehlsauflistung:

CP dd	FE dd
CP A	BF
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP H	BC
CP L	BD
CP (HL)	BE
CP (IX+dd)	DD BE dd
CP (IY+dd)	FD BE dd

3.2.7 Logische Operatoren

Es gibt beim Z80 auch logische Operatoren wie AND (und), OR (oder) und XOR (exklusiv-oder). Diese Operatoren beziehen sich alle auf den Akkumulator. Es wird bitweise logisch abgearbeitet.

3.2.7.1 AND

Der Akkumulator wird mit einem Byte Bit für Bit 'undiert'; das heißt, wenn zwei Bits gleicher Wertigkeit gesetzt sind, so wird im Ergebnis, welches wieder in den Akkumulator kommt, das Bit dieser Wertigkeit gesetzt. - Beispiel:

01100010 bin = 62 hex = 98 dez

AND

11010110 bin = D6 hex = 214 dez

ergibt

01000010 bin = 42 hex = 66 dez.

AND-Operationen setzen das Carry Flag auf Null.

Hier die Befehlsauflistung:

```
AND dd                  E6 dd
AND A                   A7
AND B                   A0
AND C                   A1
AND D                   A2
AND E                   A3
AND H                   A4
AND L                   A5
AND (HL)                A6
AND (IX+dd)             DD A6 dd
AND (IY+dd)             FD A6 dd
```

Ein Programmbeispiel:

```
16514   LD B,00         6   0
16516   LD A,13         62  19
16518   LD C,E1         14  225
16520   AND C           161
16521   LD C,A          79
16522   RET             201
```

Nach PRINT USR 16514 muß 1 herauskommen. Sie können es nachprüfen, indem Sie 225 und 19 ins Binärsystem übersetzen und von Hand 'undieren'.

3.2.7.2 OR

Der Akkumulator wird mit einem Byte Bit für Bit 'oderiert'; das heißt, wenn mindestens ein Bit gesetzt ist, so wird auch im Ergebnis, das auch wieder in den Akkumulator kommt, das Bit gesetzt.

Beispiel:

01100010 bin = 62 hex = 98 dez

OR

11010110 bin = D6 hex = 214 dez

ergibt

11110110 bin = F6 hex = 246 dez.

Die Befehle:

```
OR dd           F6 dd
OR A            B7
OR B            B0
OR C            B1
OR D            B2
OR E            B3
OR H            B4
OR L            B5
OR (HL)         B6
OR (IX+dd)      DD B6 dd
OR (IY+dd)      FD B6 dd
```

Beispiel:

```
16514   LD B,00         6   0
16516   LD A,13         62  19
16518   LD C,E1         14  225
16520   OR C            177
16521   LD C,A          79
16522   RET             201
```

Wenn man das vorige Programm noch im Rechner hat, dann braucht man nur POKE 16520,177 einzugeben, um das Programm in die obige Form zu bringen. Nach PRINT USR 16514 sollte 243 herauskommen, was man hier auch schriftlich am besten überprüfen kann.

3.2.7.3 XOR

Der Akkumulator wird mit einem Byte Bit für Bit 'exklusiv oderiert'; das heißt, wenn genau ein Bit gesetzt ist, so wird auch im Ergebnis das Bit gesetzt. - Beispiel:

01100010 bin = 62 hex = 98 dez

XOR

11010110 bin = D6 hex = 214 dez

ergibt

10110100 bin = B4 hex = 180 dez.

Die Befehlsauflistung:

```
XOR dd              EE dd
XOR A               AF
XOR B               A8
XOR C               A9
XOR D               AA
XOR E               AB
XOR H               AC
XOR L               AD
XOR (HL)            AE
XOR (IX+dd)         DD AE dd
XOR (IY+dd)         FD AE dd
```

Der Leser wird jetzt wohl selbst imstande sein, sich eigenes kleines Maschinencodeprogramm als Beispiel zu schreiben.

3.2.8 Sprungbefehle

Die Sprungbefehle im Maschinencode funktionieren genauso wie GOTO in BASIC, nur folgt keine Zeilennummer, sondern die Adresse, ab der weitergearbeitet werden soll. Man kann die Sprungbefehle in zwei Gruppen einteilen:

- die absoluten Sprünge: mit ihnen kann man zu **jeder** Speicheradresse springen;
- die relativen Sprünge: mit ihnen kann man in einem bestimmten Bereich vorwärts und rückwärts springen.

3.2.8.1 Absolute Sprünge

Es gibt einen Befehl, bei dem auf jeden Fall zu der danach folgenden Adresse gesprungen wird, nämlich

```
JP addr             C3 addr
```

Durch diesen Befehl wird der Programmzähler einfach mit addr geladen.

Ein Beispiel:

```
16514    LD BC,F0F0          1 240 240
16517    JP 408B           195 139  64
16520    LD BC,0000          1   0   0
16523    RET               201
```

Das Programm wird mit PRINT USR 16514 gestartet, und es müßte 61680 gedruckt werden. BC wird mit 61680 geladen, und danach wird zum RET-Befehl gesprungen. Damit wird der Befehl, der BC mit 0 laden soll, übergangen.
Die drei relativen Sprungbefehle, die zu Adressen springen, die in Doppelregistern stehen, sind

```
JP (HL)          E9
JP (IX)          DD E9
JP (IY)          FD E9
```

Um dies zu verdeutlichen, hier ein Beispiel:
Angenommen, im HL-Doppelregister steht 3196 (die Einsprungadresse von LIST), so bewirkt der Befehl JP (HL) das gleiche wie JP 3196.

3.2.8.1.1 Sprünge bedingter Art

Wie schon vorher erwähnt, kann man sich auf 4 Flags beziehen, das Carry Flag, das Zero Flag, das Sign Flag und das Parity/Overflow Flag. Der Test dieser Flags wird wie folgt bezeichnet:

```
C  wenn Carry Flag gesetzt          (carry)
NC wenn Carry Flag nicht gesetzt    (no carry)
Z  wenn Zero Flag gesetzt           (zero)
NZ wenn Zero Flag nicht gesetzt     (no zero)
M  wenn Sign Flag gesetzt           (minus)
P  wenn Sign Flag nicht gesetzt     (plus)
PE wenn P/O Flag gesetzt            (parity even)
PO wenn P/O Flag nicht gesetzt      (parity odd).
```

Jetzt kann jeder Sprung mit einer Bedingung verknüpft werden, z.B.

JP Z,addr

bedeutet:

Springe nach addr, wenn das Zero Flag gesetzt ist.

Die Befehlsauflistung:

```
JP NC,addr          D2 addr
JP C,addr           DA addr
JP NZ,addr          C2 addr
JP Z,addr           CA addr
JP P,addr           F2 addr
JP M,addr           FA addr
JP PO,addr          E2 addr
JP PE,addr          EA addr
```

Nachdem Sie diese Befehle kennen, können Sie jetzt schon Schleifen und ähnliches programmieren. Hierzu ein Beispiel:

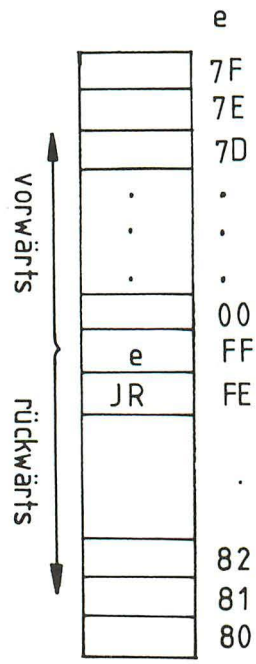
Angenommen, Sie wollen 20 x 4 in Maschinencode ausrechnen. Dazu braucht man eine Schleife, die 20mal 4 addiert, weil die Multiplikation schließlich eine wiederholte Addition ist. Ein Zähler wird auch noch benötigt, wofür wir das B-Register benutzen. Das C-Register enthält den Wert, der addiert werden soll, also 4. Nach jedem Durchlauf der Schleife wird B um eins erniedrigt; wenn B auf Null steht, wird wieder in die Schleife zurückgesprungen.

```
16514   LD B,14          6  20
16516   LD C,04         14   4
16518   LD A,00         62   0
16520   ADD A,C        129
16521   DEC B            5
16522   JP NZ,4088     194 136  64
16525   LD C,A          79
16526   RET            201
```

PRINT USR 16514 müßte den Ausdruck von 80 bewirken.

3.2.8.2 Relative Sprünge

Bei relativen Sprüngen wird nur die Sprungweite vorwärts oder rückwärts angegeben, so daß diese Befehle problemlos mitsamt dem Programm verschoben werden können, ohne umgeschrieben werden zu müssen. Außerdem spart man, verglichen mit den absoluten Sprüngen, ein Byte. Mit relativen Sprüngen kann man maximal 127 Bytes vorwärts oder 128 Bytes rückwärts springen. Diese Sprungweite wird im folgenden mit der Variablen 'e' benannt. Hier ein Diagramm, welches die Sprungmöglichkeiten aufzeigt:



3.2.8.2.1 Relative Sprünge unbedingter Art

Es gibt nur einen Befehl unbedingter Art:

JR e	18 e

3.2.8.2.1.1 DJNZ

Dieser Befehl ist ein automatischer, relativer unbedingter Sprungbefehl, der sich auf das B-Register bezieht und ausgeschrieben 'decrement B and jump if not zero' bedeutet.

DJNZ e	10 e

Mit diesem Befehl kann man das Multiplikationsprogramm vereinfachen und kürzen:

16514	LD B,14	6	20
16516	LD C,04	14	4
16518	LD A,00	62	0
16520	ADD A,C	129	
16521	DJNZ FD	16	253
16523	LD C,A	79	
16525	RET	201	

3.2.8.2.2 Relative Sprünge bedingter Art

Relative Sprünge können nur das Carry und Zero Flag testen:

JR NC,e	30 e
JR C,e	38 e
JR NZ,e	20 e
JR Z,e	28 e

3.2.9 Unterprogramm-Befehle

Im Maschinencode gibt es drei Arten von Unterprogramm-Befehlen: CALL, RET und RST. CALL ist dem BASIC-Befehl GOSUB gleichzusetzen. RET ist dann in BASIC RETURN und RST ist eine Kurzform von CALL. Jeder CALL- und RET-Befehl kann die Flags testen wie die JP-Befehle.

3.2.9.1 CALL

Wenn ein CALL ausgeführt wird, wird die Adresse, zu der wieder zurückgesprungen werden soll, auf den Stapel gebracht. Dann wird mit der Adresse, zu der durch CALL gesprungen werden soll, weitergemacht. Die Befehlsauflistung enthält ein unbedingtes CALL, der Rest bezieht sich auf die vier Flags.

CALL addr	CD addr
CALL C,addr	DC addr
CALL NC,addr	D4 addr
CALL Z,addr	CC addr
CALL NZ,addr	C4 addr
CALL M,addr	FC addr
CALL P,addr	F4 addr
CALL PE,addr	EC addr
CALL PO,addr	E4 addr

Der BASIC USR-Befehl ist ja auch ein CALL, da ja nach RET wieder zu ihm zurückgesprungen wird.

3.2.9.2 RET

Wenn ein RET ausgeführt wird, wird die letzte Adresse vom Stapel geholt und in den Programmzähler gebracht. Dies ist meistens die Adresse+1 des letzten CALL-Befehls.
Es kann auch jeder RET-Befehl eines der vier Flags abfragen, so daß z.B. mit RET PE bei einem arithmetischen Überlauf zurückgesprungen wird.

RET	C9
RET C	D8
RET NC	D0
RET Z	C8
RET NZ	C0
RET M	F8
RET P	F0
RET PE	E8
RET PO	E0

Zu diesen Befehlen braucht wohl kein Beispiel geliefert zu werden, da man die Funktion ja vom BASIC her kennt.

3.2.9.3 RST

Es gibt acht vordefinierte RST-Befehle, die einen CALL-Befehl darstellen, in dem die Adresse im Befehlsbyte integriert ist. Man spart also zwei Bytes.

RST 00	C7 = CALL 0000
RST 08	CF = CALL 0008
RST 10	D7 = CALL 0010
RST 18	DF = CALL 0018
RST 20	E7 = CALL 0020
RST 28	EF = CALL 0028
RST 30	F7 = CALL 0030
RST 38	FF = CALL 0038

Da diese Befehle alle ins ROM springen, muß man wissen, welche Routinen dort liegen, wenn man einen RST-Befehl benutzen will.

3.2.10 Stapeloperationen

Man kann den Stapel auch dazu benutzen, Daten zwischenzuspeichern; dazu braucht man Befehle, die Daten zum Stapel bringen und wieder zurückholen. Zum Auf-den-Stapel-Bringen verwendet man den Befehl PUSH, der die zwei Bytes eines Doppelregisters auf den Stapel bringt. Mit POP kann man zwei Bytes vom Stapel in ein Doppelregister laden.

3.2.10.1 PUSH

Die Befehlsauflistung:

PUSH AF	F5
PUSH BC	C5
PUSH DE	D5
PUSH HL	E5
PUSH IX	DD E5
PUSH IY	FD E5

3.2.10.2 POP

Die Befehlsauflistung:

POP AF	F1
POP BC	C1
POP DE	D1
POP HL	E1
POP IX	DD E1
POP IY	FD E1

Ein Beispielprogramm:

```
16515   LD BC,FFFF      1 255 255
16517   LD HL,0F0F     33  15  15
16518   PUSH BC       197
16519   PUSH HL       229
16520   POP BC        193
16521   POP HL        225
16522   RET           201
```

Nach PRINT USR 16514 müßte 3855 herauskommen (0F0F hex). Zuerst wird BC mit 65535 geladen, dann HL mit 3855. Nun wird BC auf den Stapel gebracht, danach HL. Der Stapel hat also als unterstes Element 3855; dieser Wert wird mit POP BC in BC geladen, danach 65535 mit POP HL in HL. Mit anderen Worten: es wurden die Inhalte der beiden Doppelregister vertauscht.

3.2.11 Rotations- und Schiebebefehle

Es gibt beim Z80 viele Rotations- und Schiebebefehle, um Bits in einem Byte nach rechts oder links zu verschieben. Diese Befehle sind vielseitig verwendbar, da sie das Byte verdoppeln, wenn nach links verschoben wird, und es halbieren, wenn nach rechts geschoben wird. Alle Befehl nutzen das Carry Bit als Überlaufbit, also als Bit 8 oder als Bit -1.

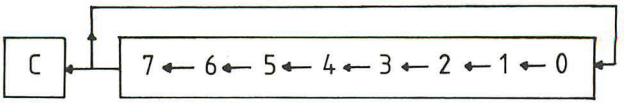
Auf der nächsten Seite werden die verschiedenen Arten von Rotations- und Schiebebefehlen grafisch veranschaulicht.

Die Befehlsauflistung der vier Akkumulatorbefehle:

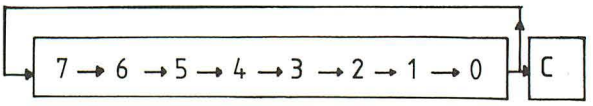
RLCA	07
RL	17
RRCA	0F
RR	1F

	RLC	RL	SLA	RRC	RR	SRA	SRL
A	CB 07	CB 17	CB 27	CB 0F	CB 1F	CB 2F	CB 3F
B	CB 00	CB 10	CB 20	CB 08	CB 18	CB 28	CB 38
C	CB 01	CB 11	CB 21	CB 09	CB 19	CB 29	CB 39
D	CB 02	CB 12	CB 22	CB 0A	CB 1A	CB 2A	CB 3A
E	CB 03	CB 13	CB 23	CB 0B	CB 1B	CB 2B	CB 3B
H	CB 04	CB 14	CB 24	CB 0C	CB 1C	CB 2C	CB 3C
L	CB 05	CB 15	CB 25	CB 0D	CB 1D	CB 2D	CB 3D
(HL)	CB 06	CB 16	CB 26	CB 0E	CB 1E	CB 2E	CB 3E
(IX+dd)	DD CB dd 06	DD CB dd 16	DD CB dd 26	DD CB dd 0E	DD CB dd 1E	DD CB dd 2E	DD CB dd 3E
(IY+dd)	FD CB dd 06	FD CB dd 16	FD CB dd 26	FD CB dd 0E	FD CB dd 1E	FD CB dd 2E	FD CB dd 3E

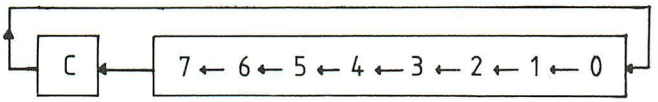
RLC / RLCA
Rotation nach links
mit Carry



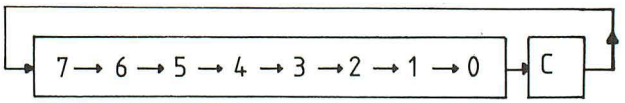
RRC/RRCA
Rotation nach rechts
mit Carry



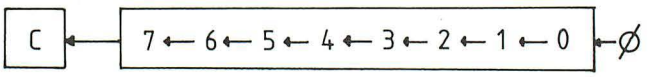
RL/RLA
Rotation nach links
durch Carry



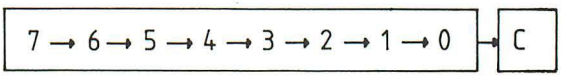
RR/RRA
Rotation nach rechts
durch Carry



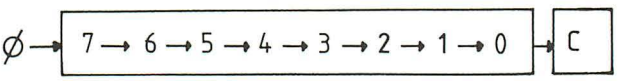
SLA
Verschieben nach
links



SRA
Verschieben nach rechts

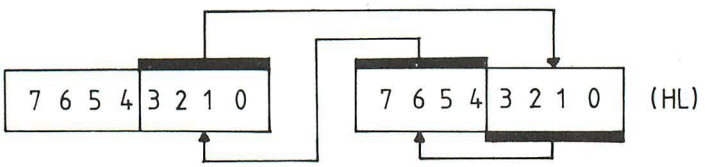


SRL
Logische Verschiebung
nach rechts



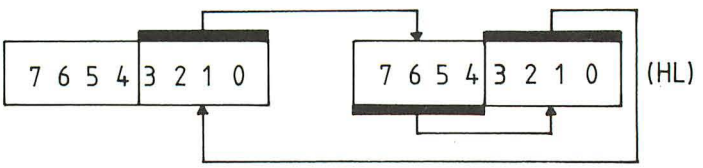
RLD

Akkumulator



RRD

Akkumulator



```
RRD     ED 67
RLD     ED 6F
```

Alle Befehle außer RLCA, RLA, RRCA und RRA beziehen sich aufs Zero, O/P und das Sign Flag. - Ein Beispiel:

```
16514   LD B,00          6   0
16516   LD C,0F         14  15
16518   SLA C          203  33
16520   RET            201
```

Zuerst wird BC mit 15 geladen, dann wird C nach links verschoben, was den Inhalt von C verdoppelt. Nach PRINT USR 16514 muß 30 gedruckt werden.

3.2.12 Einzelbitverarbeitung

Alle Einzelbitbefehle beziehen sich auf das Zero Flag. Das Zero Flag wird gesetzt, wenn das spezielle Bit Null ist, rückgesetzt, wenn das Bit 1 ist.

3.2.12.1 Setzen von Einzelbits

Mit dem SET-Befehl können Einzelbits in den Registern A, B, C, D, E, H und L und in den Speicherstellen (HL), (IX+dd) und (IY+dd) gesetzt werden.

SET	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
A	CB C7	CB CF	CB D7	CB DF	CB E7	CB EF	CB F7	CB FF
B	CB C0	CB C8	CB D0	CB D8	CB E0	CB E8	CB F0	CB F8
C	CB C1	CB C9	CB D1	CB D9	CB E1	CB E9	CB F1	CB F9
D	CB C2	CB CA	CB D2	CB DA	CB E2	CB EA	CB F2	CB FA
E	CB C3	CB CB	CB D3	CB DB	CB E3	CB EB	CB F3	CB FB
H	CB C4	CB CC	CB D4	CB DC	CB E4	CB EC	CB F4	CB FC
L	CB C5	CB CD	CB D5	CB DD	CB E5	CB ED	CB F5	CB FD
(HL)	CB C6	CB CE	CB D6	CB DE	CB E6	CB EE	CB F6	CB FE
(IX+dd)	DD CB	DD CB	DD CB	DD CB	DD CB	DD CB	DD CB	DD CB
	dd C6	dd CE	dd D6	dd DE	dd E6	dd EE	dd F6	dd FE
(IY+dd)	FD CB	FD CB	FD CB	FD CB	FD CB	FD CB	FD CB	FD CB
	dd C6	dd CE	dd D6	dd DE	dd E6	dd EE	dd F6	dd FE

3.2.12.2 Rücksetzen von Einzelbits

Mit dem RES-Befehl können Einzelbits in den bei 3.2.12.1 genannten Registern und Speicherstellen zurückgesetzt werden.

RES	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
A	CB 87	CB 8F	CB 97	CB 9F	CB A7	CB AF	CB B7	CB BF
B	CB 80	CB 88	CB 90	CB 98	CB A0	CB A8	CB B0	CB B8
C	CB 81	CB 89	CB 91	CB 99	CB A1	CB A9	CB B1	CB B9
D	CB 82	CB 8A	CB 92	CB 9A	CB A2	CB AA	CB B2	CB BA
E	CB 83	CB 8B	CB 93	CB 9B	CB A3	CB AB	CB B3	CB BB
H	CB 84	CB 8C	CB 94	CB 9C	CB A4	CB AC	CB B4	CB BC
L	CB 85	CB 8D	CB 95	CB 9D	CB A5	CB AD	CB B5	CB BD
(HL)	CB 86	CB 8E	CB 96	CB 9E	CB A6	CB AE	CB B6	CB BE
(IX+dd)	DD CB	DD CB	DD CB	DD CB	DD CB	DD CB	DD CB	DD CB
	dd 86	dd 8E	dd 96	dd 9E	dd A6	dd AE	dd B6	dd BE
(IY+dd)	FD CB	FD CB	FD CB	FD CB	FD CB	FD CB	FD CB	FD CB
	dd 86	dd 8E	dd 96	dd 9E	dd A6	dd AE	dd B6	dd BE

3.2.12.3 Testen von Einzelbits

Mit dem BIT-Befehl können Bits in den vorher genannten Registern und Speicherstellen getestet werden. Ist das entsprechende Bit Null, wird das Zero Flag gesetzt, ist es 1, so wird das Flag rückgesetzt.

BIT	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
A	CB 47	CB 4F	CB 57	CB 5F	CB 67	CB 6F	CB 77	CB 7F
B	CB 40	CB 48	CB 50	CB 58	CB 60	CB 68	CB 70	CB 78
C	CB 41	CB 49	CB 51	CB 59	CB 61	CB 69	CB 71	CB 79
D	CB 42	CB 4A	CB 52	CB 5A	CB 62	CB 6A	CB 72	CB 7A
E	CB 43	CB 4B	CB 53	CB 5B	CB 63	CB 6B	CB 73	CB 7B
H	CB 44	CB 4C	CB 54	CB 5C	CB 64	CB 6C	CB 74	CB 7C
L	CB 45	CB 4D	CB 55	CB 5D	CB 65	CB 6D	CB 75	CB 7D
(HL)	CB 46	CB 4E	CB 56	CB 5E	CB 66	CB 6E	CB 76	CB 7E
(IX+dd)	DD CB	DD CB	DD CB	DD CB	DD CB	DD CB	DD CB	DD CB
	dd 46	dd 4E	dd 56	dd 5E	dd 66	dd 6E	dd 76	dd 7E
(IY+dd)	FD CB	FD CB	FD CB	FD CB	FD CB	FD CB	FD CB	FD CB
	dd 46	dd 4E	dd 56	dd 5E	dd 66	dd 6E	dd 76	dd 7E

In folgendem Beispiel wird der BIT 7,C - Befehl verdeutlicht:

```
16514   LD BC,0000        1   0   0
16517   INC C            12
16518   BIT 7,C         203 121
16520   RET NZ          192
16521   JP 4085         195 133  64
```

Nach PRINT USR 16514 muß 128 ausgedruckt werden. C wird solange erhöht, bis Bit 7 (mit der Wertigkeit 128) gesetzt ist. Ist dies der Fall, wird ins BASIC zurückgesprungen.

3.2.13 Blockverschiebebefehle

Es gibt beim Z80 verschiedene Befehle, die es erlauben, Speicherblöcke zu verschieben. Bei diesen Befehlen steht in HL die Startadresse, in DE die Zieladresse und in BC die Länge des zu verschiebenden Blocks.

3.2.13.1 Nicht-automatische Blockverschiebebefehle

LDI	ED A0
LDD	ED A8

Bei diesen Befehlen wird ein einzelnes Byte von (HL) nach (DE) gebracht, das BC-Doppelregister wird dekrementiert. Dann werden das HL- und DE-Doppelregister bei

- LDI inkrementiert
- LDD dekrementiert.

3.2.13.2 Automatische Blockverschiebebefehle

LDIR	ED B0
LDDR	ED B8

Diese Befehle verhalten sich genauso wie die Befehle in Kapitel 3.2.13.1, nur daß der Einzelbefehl so oft wiederholt wird, bis BC Null ist. Das 'R' steht daher für **REPEAT**. Ein Beispiel hierzu ist in Kapitel 5.1 aufgeführt.

3.2.14 Blocksuchbefehle

Auch hier gibt es automatische und nicht-automatische Befehle. Bei diesen Befehlen muß im Akkumulator der Wert stehen, nach dem gesucht wird, und zwar ab (HL). BC ist die Weite, bis zu der ab (HL) gesucht werden soll.

3.2.14.1 Nicht-automatische Blocksuchbefehle

CPI	ED A1
CPD	ED A9

Es wird Byte (HL) mit A verglichen und das Zero Flag entsprechend gesetzt (siehe CP-Befehle). Dann wird BC dekrementiert und HL bei

- CPI inkrementiert
- CPD dekrementiert.

Ist BC bei Null angekommen, wird das P/O Flag rückgesetzt.

3.2.14.2 Automatische Blocksuchbefehle

CPIR	ED B1
CPDR	ED B9

Bei CPIR wird theoretisch so lange CPI ausgeführt, bis BC Null ist oder A gefunden wurde. Ist A gefunden, wird das Zero Flag gesetzt, und HL enthält die nachfolgende Adresse des gefundenen Bytes. Bei CPDR wird CPD sooft abgearbeitet, bis Bedingungen wie bei CPIR eintreten. Ist BC Null, wird das Zero Flag zurückgesetzt, ebenso das Sign und P/O Flag.

Ein Beispiel:

```
16514   LD HL,0000      33   0   0
16517   LD BC,2000       1   0  32
16520   LD A,FF         62 255
16522   CPIR           237 177
16524   LD B,H          68
16525   LD C,L          77
16526   DEC BC          11
16527   RET            201
```

Nach PRINT USR 16514 muß 3 herauskommen. Sie können es mit PRINT PEEK 3 überprüfen. Der Wert, der hier herauskommt, ist 255. Zuerst wird HL mit dem ersten möglichen Wert geladen, also 0. In BC steht die Länge des ROMs, in A der Wert, nach dem gesucht wird. Mit CPIR wird dann gesucht, bis 255 gefunden ist. Dann wird die Speicherstelle +1 in BC geladen, BC dekrementiert, damit wir die Speicherstelle, in der 255 enthalten ist, bekommen.

3.2.15 Transfer-Befehle

Es gibt zwei Befehlsarten, mit denen man externe Einrichtungen wie den Drucker, die Tastatur oder den Cassettenrecorder ansprechen kann. Man kann mit dem OUT-Befehl Daten zu diesen Geräten senden, oder mit dem IN-Befehl Daten von diesen Geräten empfangen. Wenn ein OUT-Befehl abgearbeitet wird, wird auf den Datenbus der Inhalt eines bestimmten Registers geschickt. Beim IN-Befehl wird ein Byte vom Datenbus geholt und in ein bestimmtes Register geladen.
Nun braucht man aber noch einen Wert, nämlich die Adresse, mit der ein externes Gerät angesprochen werden kann. Jedes Gerät besitzt eine bestimmte Adresse, die PORT-Adresse. Diese PORT-Adresse wird bei IN und OUT auf den Datenbus gelegt, um so festzulegen, mit welcher Einheit kommuniziert werden soll.

		Ein/Ausg.-Register	Höherwert. PORT-Adr.	Niederwert. PORT-Adr.
IN A,(dd)	DB dd	A	A	dd
IN A,(C)	ED 78	A	B	C
IN B,(C)	ED 40	B	B	C
IN C,(C)	ED 48	C	B	C
IN D,(C)	ED 50	D	B	C
IN E,(C)	ED 58	E	B	C
IN H,(C)	ED 60	H	B	C
IN L,(C)	ED 68	L	B	C
OUT (dd),A	D3 dd	A	A	dd
OUT (C),A	ED 79	A	B	C
OUT (C),B	ED 41	B	B	C
OUT (C),C	ED 49	C	B	C
OUT (C),D	ED 51	D	B	C
OUT (C),E	ED 59	E	B	C
OUT (C),H	ED 61	H	B	C
OUT (C),L	ED 69	L	B	C

Beispiel:

```
16514   LD B,FF          6 255
16516   OUT (FF),A     211 255
16518   DJNZ 16516      15 252
16520   RET            201
```

3.2.15.1 Block-Transfer-Befehle

Diese Befehle werden hier nur kurz aufgelistet und nicht besonders erklärt, da sie fast nur im Betrieb mit Disketten verwendet werden:

INI	ED A2
INIR	ED B2
IND	ED AA
INDR	ED BA
OUTI	ED A3
OUTIR	ED B3
OUTD	ED AB
OUTDR	ED 8B

Auch hier steht am Anfang entweder IN oder OUT, dann I oder D für Inkrement oder Dekrement, eventuell noch R für REPEAT (automatische Befehle).

Dieses Programm sollte in einer Schleife laufen:

```
100 FOR A=1 TO 200
110 RAND USR 16514
120 NEXT A
```

und mit GOTO 100 gestartet werden. Es produziert einen Brummton am MIC-Ausgang.

3.2.16 Interrupt-Befehle

Der Z80 kann 'unterbrochen' werden. Das heißt, er kann durch ein 'Interrupt' seinen Programmablauf unterbrechen, ein anderes Programm zwischendurch abarbeiten, und danach sein unterbrochenes Programm weiter fortführen. Es gibt beim Z80 zwei Leitungen, die im ZX81 Interrupts bewirken: Die NMI- und die INT-Leitung.
Wenn die NMI- (nicht maskierbarer Interrupt) Leitung aktiviert wird, hört der Z80 auf, sein aktuelles Programm abzuarbeiten, speichert den Programmzähler auf dem Stapel ab, damit er weiß, wo er nach Abarbeiten des Interrupts weitermachen soll, und lädt den Programmzähler mit 66 hex, die Startadresse der Bildschirmgenerierung im SLOW-Modus.

Die INT- (maskierbarer Interrupt) Leitung arbeitet genauso wie die NMI, außer daß nicht 66 hex in den Programmzähler geladen wird, und daß die Leitung an- und abgeschaltet werden kann.

EI FB

bewirkt, daß die INT-Leitung wirksam ist.

DI F3

schaltet die INT-Leitung ab.
Wenn ein INT vorkommt, wird die INT-Leitung automatisch abgeschaltet.
Es gibt drei verschiedene Modi für den maskierbaren Interrupt, sie werden angewählt mit:

IM 0	ED 46
IM 1	ED 56
IM 2	ED 5E.

Modus 0 bewirkt, daß ein externes Gerät ein Byte auf den Datenbus schickt, wenn ein Interrupt vorliegt, das anzeigt, welche RST-Adresse in den Programmzähler geladen werden soll.
Modus 1 bingt 38 hex zum Programmzähler.
Modus 2 bringt eine Zeigeradresse in den Programmzähler, die sich aus dem Inhalt des I-Registers und des Bytes zusammensetzt, welches sich auf dem Datenbus befindet.
Normalerweise wird mit RET von einem Interrupt zurückgesprungen, doch gibt es noch zwei spezielle Interrupt-Return-Befehle:

RETI	ED 4D
RETN	ED 45

Mit RETI wird der maskierbare Interrupt beim Zurückspringen von einem maskierbaren Interrupt wieder eingeschaltet. Springt man von einem NMI zurück und möchte den maskierbaren Interrupt einschalten, benutzt man RETN.

Es gibt einen Befehl, der auf ein Interrupt wartet und solange NOPs ausführt, bis dieser eintrifft, nämlich

HALT 76

Beim ZX81 wird HALT zum Bildschirmgenerieren im SLOW-Modus gebraucht. Der Z80 führt NOPs aus, bis das R-Register die 32 Zeichen einer Zeile gezählt hat und ein NMI erscheint.
Hierzu ein Beispiel, das man zuerst im SLOW-Modus laufen lassen sollte.

```
16514   HALT        118
16515   RET         201
```

Es wird mit RAND USR 16514 gestartet.
Im SLOW-Modus müßte sofort 0/0 erscheinen; dies zeigt an, daß Interrupts vorkommen. Läßt man das Programm in FAST laufen, ist das Programm verloren, da es in diesem Modus kein Interrupt gibt, welcher den Z80 vom HALT zurückkehren ließe.

3.2.17 Sonstige Befehle für den Akkumulator

```
NEG         ED 44
CPL         2F
DAA         27
```

Mit NEG erhält man das Zweierkomplement des Akkumulators.
Mit CPL komplementiert man den Akkumulator, d.h. jedes gesetzte Bit wird rückgesetzt und umgekehrt. - Beispiel:

```
A     10010101 bin = 149 dez
        CPL
A     01101010 bin = 106 dez
```

DAA bringt den Akkumulator in ein dezimales Format, so daß Bits 3 - 0 die Einserstelle darstellt, Bits 7 - 4 die Zehner. Jeder Nibble kann nun Werte von 0 (0000 bin) bis 9 (1001 bin) darstellen.
Hierzu auch ein Beispiel:

```
A       00100111 bin = 39 dez
        DAA
A       00111001     = 39 dez
        (30) (9)
```

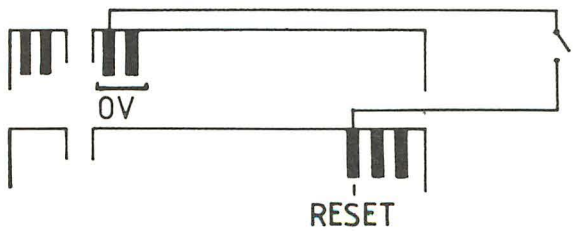
4. Maschinencode in BASIC

Nun stellt sich die Frage: Wie bringe ich Maschinencode in BASIC unter? Wenn man z.B. ein Maschinencodeprogramm, welches 2000 Bytes lang ist, in ein REM setzen möchte, so wäre es eine Lebensaufgabe, diese 2000 Bytes hinter einem REM einzugeben, da man die letzten 1000 Bytes nicht mehr sieht. Man könnte z.B. 10 REMs mit je 200 Bytes eingeben, doch dann kann man den Maschinencode nicht mehr der Reihe nach eingeben, da man auf die Zeilennummer usw. Rücksicht nehmen muß. Was bleibt noch?
Der RAMTOP-Bereich! Der RAMTOP ist das Ende des Speichers, dessen Adresse in den Speicherstellen 16388 und 16389 steht. Diese Adresse können Sie ganz leicht herausfinden, indem Sie PRINT PEEK 16388 + 256 x PEEK 16389 drücken, wobei 17408 herauskommen müßte. (Wenn Sie sich fragen, wie man zu dieser Anweisung kommt, dann stellen Sie sich vor, daß diese beiden RAMTOP-Adressen ein Doppelregister darstellen, wobei 16388 das niederwertige und 16389 das höherwertige Register ist) Die Systemvariable RAMTOP zeigt dem Computer, wieviel Speicherplatz er zur Verfügung hat. Sie aber haben die Möglichkeit, den RAMTOP zu verändern, indem Sie in diese beiden Adressen andere Zahlen hineinpoken. Nehmen wir an, Ihr Maschinencodeprogramm ist 30 Bytes lang, so daß der RAMTOP den Wert 17378 = 226 + 256 x 67 haben muß. Also drücken Sie

```
POKE 16388,226
POKE 16389,67
NEW
```

NEW hat die Funktion, daß der RAMTOP sozusagen hochgedrückt wird, indem NEW alles bis zu dem jetzt neuen RAMTOP löscht (NEW orientiert sich immer am RAMTOP). In diesem jetzt für Sie freien Speicherplatz kann Ihr Maschinencodeprogramm also auch nicht durch NEW verlorengehen. Sie können also nun Zeile 10 im obigen BASIC-Programm ersetzen durch 10 FOR A = 17378 TO 17407.

Nur läßt sich der RAMTOP nicht problemlos SAVEn. Man muß das Ganze in einen String packen, SAVEn und nach dem Laden wieder zurückpoken. Dies ist normalerweise die beste Lösung für lange Maschinencodeprogramme. Man kann ein Maschinencodeprogramm auch, wenn es fertig ist, in den freien Speicherbereich, im Handbuch SPARE genannt, legen, doch zu leicht kann es vom Stack oder vom Calculator überschrieben werden.
Für Besitzer eines 64K-RAMs bieten sich die 8K zwischen dem ROM und dem Beginn der Systemvariablen an, vor allen Dingen braucht man RAMTOP nicht zu verschieben, und der Bereich wird durch NEW nicht gelöscht. So kann man z.B. ein Maschinencodeprogramm in diesem Bereich testen, wenn z.B. eine Endlosschleife programmiert wurde, zieht man nicht den Stecker heraus, sondern benutzt einen vorher angebrachten RESET-Taster, welcher nur das BASIC-Programm löscht, nicht das Maschinencodeprogramm. Um ein RESET zu erzeugen, braucht man bloß hinten am EDGE-Connector RESET mit 0V zu verbinden:



5. Programmbeispiele

5.1 SCROLL

Dieses SCROLL arbeitet wesentlich schneller als das Sinclair-SCROLL, auch bei CLS; es sollte nicht gemischt mit dem Sinclair-SCROLL benutzt werden. Falls also das Hilfsprogramm zum Einpoken benutzt wird, bitte vor dem Starten des Maschinencodeprogramms Zeile 20 SCROLL löschen (s.a. 5.2).

Start: RAND USR 16514

16514	LD HL,(4006)	42	12	64	Das Programm lädt D-File in HL.
16517	INC HL	35			Anschließend wird HL um eins erhöht, es zeigt somit auf das erste Bildschirmelement.
16518	LD D,H	84			Der Inhalt von HL wird nach DE geladen.
16519	LD E,L	93			
16520	LD BC,0021	1	33	0	BC wird mit 33, der Zeilenlänge + N/L, geladen.
16523	ADD HL,BC	9			BC wird zu HL addiert. HL zeigt auf die 2. Zeile. DE zeigt auf die 1. Zeile.
16524	LD BC,02D6	1	214	2	BC wird mit 726 (22 Zeilen x 33 Spalten) geladen.
16527	LDIR	237	176		LDIR verschiebt den Block.
16529	RET	201			Rückkehr ins BASIC.

5.2 CLS

Dieses CLS arbeitet schneller als das Sinclair-CLS; es darf jedoch nie mit dem Sinclair-SCROLL benutzt werden!
Das Programm beginnt bei 16529 und steht somit direkt hinter dem SCROLL-Programm. Zur Eingabe müssen Sie Zeile 1 des Maschinencode-Eingabeprogramms um zwei Leerzeichen erweitern, und außerdem muß Zeile 10 folgendermaßen aussehen:

10 FOR A = 16530 TO 16546

Start: RAND USR 16530

16530	LD HL,(4006)	42	12	64	Das Programm lädt D-File in HL.
16533	LD C,16	14	22		B wird mit 22 (Zeilenanzahl) geladen.
16535	LD B,20	6	32		C wird mit 32 (Spaltenanzahl) geladen.
16537	INC HL	35			Anschließend wird HL um eins erhöht (ähnlich wie beim SCROLL-Programm).
16538	LD (HL),0	54	0		Das von HL angezeigte Bildschirmelement wird auf 0 gesetzt.
16540	DJNZ 4098	16	251		Rücksprung für die nächste Bildschirmspeicherstelle.
16542	INC HL	35			Erhöht HL um eins (N/L).
16543	DEC C	13			Erniedrigt C um eins, damit die nächste Zeile bearbeitet wird.
16544	JR NZ,4096	32	245		Rücksprung zur Bearbeitung der nächsten Zeile.
16546	RET	201			Rückkehr ins BASIC.

Um den Bildschirm mit anderen Zeichen zu füllen, muß eingegeben werden: POKE 16539, Code des Zeichens.

5.3 RAMTOP-Verschiebeprogramm

Dieses Programm verschiebt RAMTOP ohne NEW. Dies hat den Vorteil, daß das BASIC-Programm im Speicher erhalten bleibt. In die Adressen 16526 und 16527 wird die Länge gepokt, um die nach unten verschoben wird, in unserem Fall um 100. Bitte achten Sie darauf, daß in 16526 das niederwertige Byte gepokt wird!

Start: RAND USR 16515

```
16514  LD HL,(4004)   42   4  64   RAMTOP in HL
16517  SBC HL,SP     237 114       Blocklänge wird ermittelt und
16519  LD B,H         68           in BC kopiert.
16520  LD C,L         77
16521  LD (4091),SP  237 115       SP retten
                      145  64
16525  LD DE,0064     17 100   0   Verschieben um xxxx Bytes
16528  LD HL,0000     33   0   0   Hier steht nachher SP
16531  SBC HL,DE     237  82       Wohin verschoben wird
16533  LD (4091),HL   34 145  64   HL retten
16536  ADD HL,DE      25           Startadresse errechnen
16537  LD DE,(4091)  237  91       Ziel holen
                      145  64
16541  LDIR          237 176       Block verschieben
16545  LD (4004),DE  237  83       RAMTOP erneuern
                        4  64
16547  LD HL,(4002)   42   2  64   )
16550  LD DE,(402A)  237  91       }ERR-SP-korrigieren
                      142  64      }
16554  SBC HL,DE      25           )
16555  LD (4002),HL   34   2  64
16558  LD SP,(4091)  237 123       SP korrigieren
16562  RET           Zurück ins BASIC.
```

Quellen:

Z80-Assembler-Handbuch, Hofacker.
Ian Logan: Understanding your ZX81 ROM, Melbourne House.
Ian Logan: ROM-Disassembly, part A, part B, Melbourne House.
C. Lorenz: Programmieren in Maschinensprache mit Z80, Hofacker.

* ROM-EINSPRUNGADRESSEN *

* ALPHABETISCH *

AB (HL) WERDEN BC LEERZEICHEN GEBRACHT	2462
ABS	7391
ADDITION	5974
ADRESSEN-TABELLE	6436
ARCCOS	7637
ARCSIN	7621
ARCTAN	7543
AUSGABE EINER BASIC-ZEILE	1861
AUSGABE EINER FLIESSKOMMAZAHL	5599
AUSGABE EINES CHARACTERS	16
BC-LEERZEICHEN	48
BILDSCHIRM-ENDE	3095
BREAK GEDRUECKT?	982
BREAK-1	3916
BRINGT FLIESSKOMMAZAHL ENTWEDER ZUM STAPEL ODER VOM STAPEL ZUM CALC.	6647
BRINGT WERTE VON A-E NACH STAPEL	4806
CALCULATOR	6558
CALCULATOR-ENDE	43
CALCULATOR-TABELLEN	6422
CHR$	7056
CLEAR	5282
CLS	3118
CODE	7175
CONT	3708
COPY	2153
COS	7487
CURSOR-LINKS	1398
CURSOR-RAUF	1439
CURSOR-RECHTS	1407
CURSOR-RUNTER	1108
CURSOR-TASTEN-TABELLE	1378
DE=(DE+1)	4867
DEBOUNCE	3921
DIM	5133
DISPLAY-ROUTINE	519
DIVISION	6275
DRUCKT DRUCKERPUFFER BIS NEWLINE	5836
E-LINE-NUMMER	3187
EDIT	1476
EIN LEERZEICHEN	2459
ERFASST CHARACTER	24
ERFASST NAECHSTEN CHARACTER	32
ERFASST ZEILENZAHL	1463
ERHOEHE ARBEITSRAUM	5260
ERHOEHE CHARACTER-ADRESSE UM EINS	73
ERNIEDRIGT BERG UM EINS	7192
ERROR-RESTART	8
ERZEUGT EINE VARIABLE: A$+B$	7011
ERZEUGT LEEREN CALC.-STAPEL BEI HL	5338
ERZEUGT SERIEN DES CHEBYSHEV POLINOMS	7306
EXP	7260
FAST	3881
FAST-MODUS	743

FEHLER: ARITHMETISCHER UEBERLAUF 6273
FEHLER: AUSSERHALB RAM 3795
FEHLER: INPUT ALS DIREKT-KOMMANDO 3879
FEHLER: INTEGER AUSSERHALB BEREICH 7122
FEHLER: NAME NICHT VORHANDEN 756
FEHLER: NEXT OHNE FOR 3672
FEHLER: RETURN OHNE GOSUB 3813
FLIESSKOMMA-CALCULATOR 40
FLIESSKOMMAZAHL AUF STAPEL 5474
FLIESSKOMMAZAHL NACH A 5585
FLIESSKOMMAZAHL NACH BC 5522
FOR 3513
FORMT X AUS SIN X UND COS X IN EINEN WERT UM 7449
FUEHRT EINZEL-OPERATIONEN AUS 6629
FUNCTION 1503
GIBT A IN EINER FLISSKOMMAZAHL AN 5409
GIBT BC IN EINER FLIESSKOMMAZAHL A 5412
GOSUB 3765
GOTO 3713
HAELT ZWEI ZEILEN FREI 5345
HL ZEIGT AUF STKEND-5, DE AUF STKEND 7045
HL=HL*DE 4873
HOLT ZWEI ZAHLEN VOM STAPEL 5883
IF 3547
INITIALISIERUNG 997
INKEY$ 4000
INPUT 3817
INT 7239
INTEGER AUSSERHALB BEREICH 3805
INTEGER-GEFUNDEN 3799
INTERRUPT 56
K- ODER L-MODUS DES CURSORS 1335
KOMMANDO-KLASSE 0 3373
KOMMANDO-KLASSE 1 3388
KOMMANDO-KLASSE 2 3435
KOMMANDO-KLASSE 3 3368
KOMMANDO-KLASSE 4 3461
KOMMANDO-KLASSE 5 3374
KOMMANDO-KLASSE 6 3474
KOMMANDO-KLASSEN-TABELLE 3350
KONSTANTEN-TABELLE 6422
KONTROLL-ENDE 3357
KOPIERT DIE EDIT-ZEILE 1135
L WIRD MIT >DATA-BYTE< GELADEN 86
LEN 7186
LET 4901
LIST 1836
LN 7386
LOAD 832
LOAD/SAVE-VORBEREITUNG 508
LOC.-ADDR 2328
LOESCHE ARBEITSRAUM 5335
LOESCHE ARBEITSRAUM BIS HL 5323
LOESCHE DEN DRUCKER-PUFFER 2274
LOESCHE EIN ZEICHEN 1372
LPRINT 3275
LPRINTE EIN ZEICHEN 2129
MODULUS 7224
MULTIPLIKATION 6087
NAECHSTE ZEILE ODER VARIABLE 2546
NEGIERT LETZTEN WERT DES CALC.STAPELS 7377
NEW 963
NEWLINE 1548
NEXT 3630
NEXT-SCHLEIFE 3678
NMI 102
NOCH EIN ZEICHEN ZUR E-LINE 1318

NOT 7382
NUMBER 1972
OR 7406
PAUSE 3896
PEEK 7359
PI 3983
PLOT X-KOORDINATE IN B, Y IN C 3039
POKE 3730
POTENZIEREN 7651
PRINT 3279
PRINT AT ZEILE IN B, SPALTE IN C 2293
PRINT TAB 3326
PRINT-UNTERROUTINE 2012
PRIORITAETSTABELLE 4371
PROGRAMM-NAME 984
RAND 3692
REPORT + ZEILENNUMMER DER AUSGABE 3224
RET BEI: LETZTER WERT KLEINER 0 7388
RET MIT C-FLAG WENN A=GUELTIGER BUCHSTABE 5334
RET MIT C-FLAG WENN A=GUELTIGES ALPHANUMERISCHES ZEICHEN 5330
RETURN 3800
RND 3932
RUBOUT 1419
RUECKSETZEN 3165
RUFT SYNTAX-Z AUF 3269
SAVE 758
SCHREIBE SCHLUESSELWOERTER AUS 2379
SCROLL 3086
SGN 7392
SIN 7500
SLOW 3889
SLOW-VORBEREITUNG 111
SORTIER-MODUS 1247
SORTIEREN DER CURSORTASTE 1323
SPEICHERPUNKT NACH MEMBOT 5312
SQR 7644
STAPEL NACH A 3074
STAPEL NACH BC 3061
START 0
STOP 3292
STR$ 7126
SUBTRAKTION 5965
SUCHE ERSTES ZEICHEN AUS E-LINE 1427
SUCHE NAECHSTEN NUMERISCEN AUSDRUCK 3928
SUCHE VARIABLEN 4384
SYNTAX-TABELLE 3113
SYNTAX-Z 3542
TAN 7535
TASTATUR-DECODIERUNG 1981
TASTATUR-SCANNING 699
TASTEN-TABELLE 126
TAUSCHT ZWEI ZAHLEN AUF DEM CALC.-STAPEL AUS 7283
TEST NACH STAPEL-ENDE 3781
TESTET 5 BYTES FUER FLIESSKOMMAZAHL 6636
UMWADLUNG IN FLIESSKOMMAZAHLEN 5341
UNPLOT X-KOORDINATE IN B, Y IN C 3039
UNTERSCHEIDE HL UND DE 3095
USR 7366
VAL 7125
VERAENDERT ALLE ZEIGER 2525
VERGLEICHS-OPERATOREN 6916
VERGLEICHT ZEILENNUMMERN 2542
VOM STAPEL:5-BYTE-FLIESSKOMMAZAHL, STRING-PARAMETER 5116
VORBEREITEN DES BASIC-LISTINGS 1049
VORBEREITUNG AUF ' / UND * 6077
VORBEREITUNG ZUR ADDITION 5852
WARTET AUF TASTENDRUCK 1217
WIEVIEL RAM? 971
ZAHL-NACH-STK 3484

```
ZEICHENCODIERUNGSTABELLE                    7680
ZEILEN-NR.-ADDR                             2520
```

* ROM-EINSPRUNGADRESSEN *

* NUMERISCH *

0	START
8	ERROR-RESTART
16	AUSGABE EINES CHARACTERS
24	ERFASST CHARACTER
32	ERFASST NAECHSTEN CHARACTER
40	FLIESSKOMMA-CALCULATOR
43	CALCULATOR-ENDE
48	BC-LEERZEICHEN
56	INTERRUPT
73	ERHOEHE CHARACTER-ADRESSE UM EINS
86	L WIRD MIT >DATA-BYTE< GELADEN
102	NMI
111	SLOW-VORBEREITUNG
126	TASTEN-TABELLE
508	LOAD/SAVE-VORBEREITUNG
519	DISPLAY-ROUTINE
699	TASTATUR-SCANNING
743	FAST-MODUS
756	FEHLER: NAME NICHT VORHANDEN
758	SAVE
832	LOAD
963	NEW
971	WIEVIEL RAM?
982	BREAK GEDRUECKT?
984	PROGRAMM-NAME
997	INITIALISIERUNG
1049	VORBEREITEN DES BASIC-LISTINGS
1108	CURSOR-RUNTER
1135	KOPIERT DIE EDIT-ZEILE
1217	WARTET AUF TASTENDRUCK
1247	SORTIER-MODUS
1318	NOCH EIN ZEICHEN ZUR E-LINE
1323	SORTIEREN DER CURSORTASTE
1335	K- ODER L-MODUS DES CURSORS
1372	LOESCHE EIN ZEICHEN
1378	CURSOR-TASTEN-TABELLE
1398	CURSOR-LINKS
1407	CURSOR-RECHTS
1419	RUBOUT
1427	SUCHE ERSTES ZEICHEN AUS E-LINE
1439	CURSOR-RAUF
1463	ERFASST ZEILENZAHL
1476	EDIT
1503	FUNCTION
1548	NEWLINE
1836	LIST
1861	AUSGABE EINER BASIC-ZEILE
1972	NUMBER
1981	TASTATUR-DECODIERUNG
2012	PRINT-UNTERROUTINE
2129	LPRINTE EIN ZEICHEN
2153	COPY
2274	LOESCHE DEN DRUCKER-PUFFER
2293	PRINT AT
2328	LOC.-ADDR
2379	SCHREIBE SCHLUESSELWOERTER AUS
2459	EIN LEERZEICHEN

```
2462  AB (HL) WERDEN BC LEERZEICHEN GEBRACHT
2520  ZEILEN-NR.-ADDR
2525  VERAENDERT ALLE ZEIGER
2542  VERGLEICHT ZEILENNUMMERN
2546  NAECHSTE ZEILE ODER VARIABLE
3039  PLOT + UNPLOT
3061  STAPEL NACH BC
3074  STAPEL NACH A
3086  SCROLL
3095  BILDSCHIRM-ENDE
3095  UNTERSCHEIDE HL UND DE
3113  SYNTAX-TABELLE
3118  CLS
3165  RUECKSETZEN
3187  E-LINE-NUMMER
3224  REPORT + ZEILENNUMMER DER AUSGABE
3262  ZEILEN-SCANNING
3269  RUFT SYNTAX-Z AUF
3275  LPRINT
3279  PRINT
3292  STOP
3326  PRINT TAB
3350  KOMMANDO-KLASSEN-TABELLE
3357  KONTROLL-ENDE
3368  KOMMANDO-KLASSE 3
3373  KOMMANDO-KLASSE 0
3374  KOMMANDO-KLASSE 5
3388  KOMMANDO-KLASSE 1
3435  KOMMANDO-KLASSE 2
3461  KOMMANDO-KLASSE 4
3474  KOMMANDO-KLASSE 6
3484  ZAHL-NACH-STK
3513  FOR
3542  SYNTAX-Z
3547  IF
3630  NEXT
3672  FEHLER: NEXT OHNE FOR
3678  NEXT-SCHLEIFE
3692  RAND
3708  CONT
3713  GOTO
3730  POKE
3765  GOSUB
3781  TEST NACH STAPEL-ENDE
3795  FEHLER: AUSSERHALB RAM
3799  INTEGER-GEFUNDEN
3800  RETURN
3805  INTEGER AUSSERHALB BEREICH
3813  FEHLER: RETURN OHNE GOSUB
3817  INPUT
3879  FEHLER: INPUT ALS DIREKT-KOMMANDO
3881  FAST
3889  SLOW
3896  PAUSE
3916  BREAK-1
3921  DEBOUNCE
3928  SUCHE NAECHSTEN NUMERISCEN AUSDRUCK
```

```
3932  RND
3983  PI
4000  INKEY$
4371  PRIORITAETSTABELLE
4384  SUCHE VARIABLEN
4806  BRINGT WERTE VON A-E NACH STAPEL
4867  DE=(DE+1)
4873  HL=HL*DE
4901  LET
5116  VOM STAPEL:5-BYTE-FLIESSKOMMAZAHL, STRING-PARAMETER
5133  DIM
5260  ERHOEHE ARBEITSRAUM
5282  CLEAR
5312  SPEICHERPUNKT NACH MEMBOT
5323  LOESCHE ARBEITSRAUM BIS HL
5330  RET MIT C-FLAG WENN A=GUELTIGES ALPHANUMERISCHES ZEICHEN
5334  RET MIT C-FLAG WENN A=GUELTIGER BUCHSTABE
5335  LOESCHE ARBEITSRAUM
5338  ERZEUGT LEEREN CALC.-STAPEL BEI HL
5341  UMWADLUNG IN FLIESSKOMMAZAHLEN
5345  HAELT ZWEI ZEILEN FREI
5409  GIBT A IN EINER FLISSKOMMAZAHL AN
5412  GIBT BC IN EINER FLIESSKOMMAZAHL A
5474  FLIESSKOMMAZAHL AUF STAPEL
5522  FLIESSKOMMAZAHL NACH BC
5585  FLIESSKOMMAZAHL NACH A
5599  AUSGABE EINER FLIESSKOMMAZAHL
5836  DRUCKT DRUCKERPUFFER BIS NEWLINE
5852  VORBEREITUNG ZUR ADDITION
5883  HOLT ZWEI ZAHLEN VOM STAPEL
5965  SUBTRAKTION
5974  ADDITION
6077  VORBEREITUNG AUF : / UND *
6087  MULTIPLIKATION
6273  FEHLER: ARITHMETISCHER UEBERLAUF
6275  DIVISION
6422  CALCULATOR-TABELLEN
6422  KONSTANTEN-TABELLE
6436  ADRESSEN-TABELLE
6558  CALCULATOR
6629  FUEHRT EINZEL-OPERATIONEN AUS
6636  TESTET 5 BYTES FUER FLIESSKOMMAZAHL
6647  BRINGT FLIESSKOMMAZAHL ENTWEDER ZUM STAPEL ODER VOM STAPEL ZUM CALC.
6916  VERGLEICHS-OPERATOREN
7011  ERZEUGT EINE VARIABLE: A$+B$
7045  HL ZEIGT AUF STKEND-5, DE AUF STKEND
7056  CHR$
7122  FEHLER: INTEGER AUSSERHALB BEREICH
7125  VAL
7126  STR$
7175  CODE
7186  LEN
7192  ERNIEDRIGT BERG UM EINS
7224  MODULUS
7239  INT
7260  EXP
7283  TAUSCHT ZWEI ZAHLEN AUF DEM CALC.-STAPEL AUS
7306  ERZEUGT SERIEN DES CHEBYSHEV POLINOMS
7359  PEEK
7366  USR
7377  NEGIERT LETZTEN WERT DES CALC.STAPELS
7382  NOT
7386  LN
7388  RET BEI: LETZTER WERT KLEINER 0
7391  ABS
7392  SGN
7406  OR
7449  FORMT X AUS SIN X UND COS X IN EINEN WERT UM
```

```
7487  COS
7500  SIN
7535  TAN
7543  ARCTAN
7621  ARCSIN
7637  ARCCOS
7644  SQR
7651  POTENZIEREN
7680  ZEICHENCODIERUNGSTABELLE
```

* UMWANDLUNGSTABELLE *

* DEZ./HEX./BIN./2ER-KOMPL. *

0	00	00000000	64	40	01000000
1	01	00000001	65	41	01000001
2	02	00000010	66	42	01000010
3	03	00000011	67	43	01000011
4	04	00000100	68	44	01000100
5	05	00000101	69	45	01000101
6	06	00000110	70	46	01000110
7	07	00000111	71	47	01000111
8	08	00001000	72	48	01001000
9	09	00001001	73	49	01001001
10	0A	00001010	74	4A	01001010
11	0B	00001011	75	4B	01001011
12	0C	00001100	76	4C	01001100
13	0D	00001101	77	4D	01001101
14	0E	00001110	78	4E	01001110
15	0F	00001111	79	4F	01001111
16	10	00010000	80	50	01010000
17	11	00010001	81	51	01010001
18	12	00010010	82	52	01010010
19	13	00010011	83	53	01010011
20	14	00010100	84	54	01010100
21	15	00010101	85	55	01010101
22	16	00010110	86	56	01010110
23	17	00010111	87	57	01010111
24	18	00011000	88	58	01011000
25	19	00011001	89	59	01011001
26	1A	00011010	90	5A	01011010
27	1B	00011011	91	5B	01011011
28	1C	00011100	92	5C	01011100
29	1D	00011101	93	5D	01011101
30	1E	00011110	94	5E	01011110
31	1F	00011111	95	5F	01011111
32	20	00100000	96	60	01100000
33	21	00100001	97	61	01100001
34	22	00100010	98	62	01100010
35	23	00100011	99	63	01100011
36	24	00100100	100	64	01100100
37	25	00100101	101	65	01100101
38	26	00100110	102	66	01100110
39	27	00100111	103	67	01100111
40	28	00101000	104	68	01101000
41	29	00101001	105	69	01101001
42	2A	00101010	106	6A	01101010
43	2B	00101011	107	6B	01101011
44	2C	00101100	108	6C	01101100
45	2D	00101101	109	6D	01101101
46	2E	00101110	110	6E	01101110
47	2F	00101111	111	6F	01101111
48	30	00110000	112	70	01110000
49	31	00110001	113	71	01110001
50	32	00110010	114	72	01110010
51	33	00110011	115	73	01110011
52	34	00110100	116	74	01110100
53	35	00110101	117	75	01110101
54	36	00110110	118	76	01110110
55	37	00110111	119	77	01110111
56	38	00111000	120	78	01111000
57	39	00111001	121	79	01111001
58	3A	00111010	122	7A	01111010
59	3B	00111011	123	7B	01111011
60	3C	00111100	124	7C	01111100
61	3D	00111101	125	7D	01111101
62	3E	00111110	126	7E	01111110
63	3F	00111111	127	7F	01111111

* UMWANDLUNGSTABELLE HIGHER-BYTE *

00.....0	40...16384	80...32768	C0...49152
01.....256	41...16640	81...33024	C1...49408
02.....512	42...16896	82...33280	C2...49664
03.....768	43...17152	83...33536	C3...49920
04....1024	44...17408	84...33792	C4...50176
05....1280	45...17664	85...34048	C5...50432
06....1536	46...17920	86...34304	C6...50688
07....1792	47...18176	87...34560	C7...50944
08....2048	48...18432	88...34816	C8...51200
09....2304	49...18688	89...35072	C9...51456
0A....2560	4A...18944	8A...35328	CA...51712
0B....2816	4B...19200	8B...35584	CB...51968
0C....3072	4C...19456	8C...35840	CC...52224
0D....3328	4D...19712	8D...36096	CD...52480
0E....3584	4E...19968	8E...36352	CE...52736
0F....3840	4F...20224	8F...36608	CF...52992
10....4096	50...20480	90...36864	D0...53248
11....4352	51...20736	91...37120	D1...53504
12....4608	52...20992	92...37376	D2...53760
13....4864	53...21248	93...37632	D3...54016
14....5120	54...21504	94...37888	D4...54272
15....5376	55...21760	95...38144	D5...54528
16....5632	56...22016	96...38400	D6...54784
17....5888	57...22272	97...38656	D7...55040
18....6144	58...22528	98...38912	D8...55296
19....6400	59...22784	99...39168	D9...55552
1A....6656	5A...23040	9A...39424	DA...55808
1B....6912	5B...23296	9B...39680	DB...56064
1C....7168	5C...23552	9C...39936	DC...56320
1D....7424	5D...23808	9D...40192	DD...56576
1E....7680	5E...24064	9E...40448	DE...56832
1F....7936	5F...24320	9F...40704	DF...57088
20....8192	60...24576	A0...40960	E0...57344
21....8448	61...24832	A1...41216	E1...57600
22....8704	62...25088	A2...41472	E2...57856
23....8960	63...25344	A3...41728	E3...58112
24....9216	64...25600	A4...41984	E4...58368
25....9472	65...25856	A5...42240	E5...58624
26....9728	66...26112	A6...42496	E6...58880
27....9984	67...26368	A7...42752	E7...59136
28...10240	68...26624	A8...43008	E8...59392
29...10496	69...26880	A9...43264	E9...59648
2A...10752	6A...27136	AA...43520	EA...59904
2B...11008	6B...27392	AB...43776	EB...60160
2C...11264	6C...27648	AC...44032	EC...60416
2D...11520	6D...27904	AD...44288	ED...60672
2E...11776	6E...28160	AE...44544	EE...60928
2F...12032	6F...28416	AF...44800	EF...61184
30...12288	70...28672	B0...45056	F0...61440
31...12544	71...28928	B1...45312	F1...61696
32...12800	72...29184	B2...45568	F2...61952
33...13056	73...29440	B3...45824	F3...62208
34...13312	74...29696	B4...46080	F4...62464
35...13568	75...29952	B5...46336	F5...62720
36...13824	76...30208	B6...46592	F6...62976
37...14080	77...30464	B7...46848	F7...63232
38...14336	78...30720	B8...47104	F8...63488
39...14592	79...30976	B9...47360	F9...63744
3A...14848	7A...31232	BA...47616	FA...64000
3B...15104	7B...31488	BB...47872	FB...64256
3C...15360	7C...31744	BC...48128	FC...64512
3D...15616	7D...32000	BD...48384	FD...64768
3E...15872	7E...32256	BE...48640	FE...65024
3F...16128	7F...32512	BF...48896	FF...65280

128	80	10000000	-128
129	81	10000001	-127
130	82	10000010	-126
131	83	10000011	-125
132	84	10000100	-124
133	85	10000101	-123
134	86	10000110	-122
135	87	10000111	-121
136	88	10001000	-120
137	89	10001001	-119
138	8A	10001010	-118
139	8B	10001011	-117
140	8C	10001100	-116
141	8D	10001101	-115
142	8E	10001110	-114
143	8F	10001111	-113
144	90	10010000	-112
145	91	10010001	-111
146	92	10010010	-110
147	93	10010011	-109
148	94	10010100	-108
149	95	10010101	-107
150	96	10010110	-106
151	97	10010111	-105
152	98	10011000	-104
153	99	10011001	-103
154	9A	10011010	-102
155	9B	10011011	-101
156	9C	10011100	-100
157	9D	10011101	-99
158	9E	10011110	-98
159	9F	10011111	-97
160	A0	10100000	-96
161	A1	10100001	-95
162	A2	10100010	-94
163	A3	10100011	-93
164	A4	10100100	-92
165	A5	10100101	-91
166	A6	10100110	-90
167	A7	10100111	-89
168	A8	10101000	-88
169	A9	10101001	-87
170	AA	10101010	-86
171	AB	10101011	-85
172	AC	10101100	-84
173	AD	10101101	-83
174	AE	10101110	-82
175	AF	10101111	-81
176	B0	10110000	-80
177	B1	10110001	-79
178	B2	10110010	-78
179	B3	10110011	-77
180	B4	10110100	-76
181	B5	10110101	-75
182	B6	10110110	-74
183	B7	10110111	-73
184	B8	10111000	-72
185	B9	10111001	-71
186	BA	10111010	-70
187	BB	10111011	-69
188	BC	10111100	-68
189	BD	10111101	-67
190	BE	10111110	-66
191	BF	10111111	-65
192	C0	11000000	-64
193	C1	11000001	-63
194	C2	11000010	-62
195	C3	11000011	-61
196	C4	11000100	-60
197	C5	11000101	-59
198	C6	11000110	-58
199	C7	11000111	-57
200	C8	11001000	-56
201	C9	11001001	-55
202	CA	11001010	-54
203	CB	11001011	-53
204	CC	11001100	-52
205	CD	11001101	-51
206	CE	11001110	-50
207	CF	11001111	-49
208	D0	11010000	-48
209	D1	11010001	-47
210	D2	11010010	-46
211	D3	11010011	-45
212	D4	11010100	-44
213	D5	11010101	-43
214	D6	11010110	-42
215	D7	11010111	-41
216	D8	11011000	-40
217	D9	11011001	-39
218	DA	11011010	-38
219	DB	11011011	-37
220	DC	11011100	-36
221	DD	11011101	-35
222	DE	11011110	-34
223	DF	11011111	-33
224	E0	11100000	-32
225	E1	11100001	-31
226	E2	11100010	-30
227	E3	11100011	-29
228	E4	11100100	-28
229	E5	11100101	-27
230	E6	11100110	-26
231	E7	11100111	-25
232	E8	11101000	-24
233	E9	11101001	-23
234	EA	11101010	-22
235	EB	11101011	-21
236	EC	11101100	-20
237	ED	11101101	-19
238	EE	11101110	-18
239	EF	11101111	-17
240	F0	11110000	-16
241	F1	11110001	-15
242	F2	11110010	-14
243	F3	11110011	-13
244	F4	11110100	-12
245	F5	11110101	-11
246	F6	11110110	-10
247	F7	11110111	-9
248	F8	11111000	-8
249	F9	11111001	-7
250	FA	11111010	-6
251	FB	11111011	-5
252	FC	11111100	-4
253	FD	11111101	-3
254	FE	11111110	-2
255	FF	11111111	-1

* Z80 - BEFEHLSSATZ *

		nach CB hex	nach ED hex
0	NOP	RLC B	
1	LD BC,dddd	RLC C	
2	LD (BD),A	RLC D	
3	INC BC	RLC E	
4	INC B	RLC H	
5	DEC B	RLC L	
6	LD B,dd	RLC (HL)	
7	RLCA	RLC A	
8	EX AF,A'F'	RRC B	
9	ADD HL,BC	RRC C	
10	LD A,(BC)	RRC D	
11	DEC BC	RRC E	
12	INC C	RRC H	
13	DEC C	RRC L	
14	LD C,dd	RRC (HL)	
15	RRCA	RRC A	
16	DJNZ e	RL B	
17	LD DE,dddd	RL C	
18	LD (DE),A	RL D	
19	INC DE	RL E	
20	INC D	RL H	
21	DEC D	RL L	
22	LD D,dd	RL (HL)	
23	RLA	RL A	
24	JR e	RR B	
25	ADD HL,DE	RR C	
26	LD A,(DE)	RR D	
27	DEC DE	RR E	
28	INC E	RR H	
29	DEC E	RR L	
30	LD E,dd	RR (HL)	
31	RRA	RR A	
32	JR NZ,e	SLA B	
33	LD HL,dddd	SLA C	
34	LD (dddd),HL	SLA D	
35	INC HL	SLA E	
36	INC H	SLA H	
37	DEC H	SLA L	
38	LD H,dd	SLA (HL)	
39	DAA	SLA A	
40	JR Z,e	SRA B	
41	ADD HL,HL	SRA C	
42	LD HL,(dddd)	SRA D	
43	DEC HL	SRA E	
44	INC L	SRA H	
45	DEC L	SRA L	
46	LD L,dd	SRA (HL)	
47	CPL	SRA A	
48	JR NC,e		
49	LD SP,dddd		
50	LD(dddd),A		
51	INC SP		
52	INC (HL)		
53	DEC (HL)		
54	LD (HL),dd		
55	SCF		
56	JR C,e	SRL B	
57	ADD HL,SP	SRL C	
58	LD A,(dddd)	SRL D	
59	DEC SP	SRL E	
60	INC A	SRL H	
61	DEC A	SRL L	
62	LD A,dd	SRL (HL)	
63	CCF	SRL A	

		nach CB hex	nach ED hex
64	LD B,B	BIT 0,B	IN B,(C)
65	LD B,C	BIT 0,C	OUT (C),B
66	LD B,D	BIT 0,D	SBC HL,BC
67	LD B,E	BIT 0,E	LD (dddd),BC
68	LD B,H	BIT 0,H	NEG
69	LD B,L	BIT 0,L	RETN
70	LD B,(HL)	BIT 0,(HL)	IM 0
71	LD B,A	BIT 0,A	LD I,A
72	LD C,B	BIT 1,B	IN C,(C)
73	LD C,C	BIT 1,C	OUT (C),C
74	LD C,D	BIT 1,D	ADC HL,BC
75	LD C,E	BIT 1,E	LD BC,(dddd)
76	LD C,H	BIT 1,H	RETI
77	LD C,L	BIT 1,L	
78	LD C,(HL)	BIT 1,(HL)	
79	LD C,A	BIT 1,A	LD R,A
80	LD D,B	BIT 2,B	IN D,(C)
81	LD D,C	BIT 2,C	OUT (C),D
82	LD D,D	BIT 2,D	SBC HL,DE
83	LD D,E	BIT 2,E	LD (dddd),DE
84	LD D,H	BIT 2,H	
85	LD D,L	BIT 2,L	
86	LD D,(HL)	BIT 2,(HL)	IM 1
87	LD D,A	BIT 2,A	LD A,I
88	LD E,B	BIT 3,B	IN E,(C)
89	LD E,C	BIT 3,C	OUT (C),E
90	LD E,D	BIT 3,D	ADC HL,DE
91	LD E,E	BIT 3,E	LD DE,(dddd)
92	LD E,H	BIT 3,H	
93	LD E,L	BIT 3,L	
94	LD E,(HL)	BIT 3,(HL)	IM 2
95	LD E,A	BIT 3,A	LD A,R
96	LD H,B	BIT 4,B	IN H,(C)
97	LD H,C	BIT 4,C	OUT (C),H
98	LD H,D	BIT 4,D	SBC HL,HL
99	LD H,E	BIT 4,E	LD (dddd),HL
100	LD H,H	BIT 4,H	
101	LD H,L	BIT 4,L	
102	LD H,(HL)	BIT 4,(HL)	
103	LD H,A	BIT 4,A	RRD
104	LD L,B	BIT 5,B	IN L,(C)
105	LD L,C	BIT 5,C	OUT (C),L
106	LD L,D	BIT 5,D	ADC HL,HL
107	LD L,E	BIT 5,E	LD DE,(dddd)
108	LD L,H	BIT 5,H	
109	LD L,L	BIT 5,L	
110	LD L,(HL)	BIT 5,(HL)	
111	LD L,A	BIT 5,A	RLD
112	LD (HL),B	BIT 6,B	
113	LD (HL),C	BIT 6,C	
114	LD (HL),D	BIT 6,D	SBC HL,SP
115	LD (HL),E	BIT 6,E	LD (dddd),SP
116	LD (HL),H	BIT 6,H	
117	LD (HL),L	BIT 6,L	
118	HALT	BIT 6,(HL)	
119	LD (HL),A	BIT 6,A	
120	LD A,B	BIT 7,B	IN A,(C)
121	LD A,C	BIT 7,C	OUT (C),A
122	LD A,D	BIT 7,D	ADC HL,SP
123	LD A,E	BIT 7,E	LD SP,(dddd)
124	LD A,H	BIT 7,H	
125	LD A,L	BIT 7,L	
126	LD A,(HL)	BIT 7,(HL)	
127	LD A,A	BIT 7,A	

		nach CB hex	nach ED hex
128	ADD A,B	RES 0,B	
129	ADD A,C	RES 0,C	
130	ADD A,D	RES 0,D	
131	ADD A,E	RES 0,E	
132	ADD A,H	RES 0,H	
133	ADD A,L	RES 0,L	
134	ADD A,(HL)	RES 0,(HL)	
135	ADD A,A	RES 0,A	
136	ADC A,B	RES 1,B	
137	ADC A,C	RES 1,C	
138	ADC A,D	RES 1,D	
139	ADC A,E	RES 1,E	
140	ADC A,H	RES 1,H	
141	ADC A,L	RES 1,L	
142	ADC A,(HL)	RES 1,(HL)	
143	ADC A,A	RES 1,A	
144	SUB B	RES 2,B	
145	SUB C	RES 2,C	
146	SUB D	RES 2,D	
147	SUB E	RES 2,E	
148	SUB H	RES 2,H	
149	SUB L	RES 2,L	
150	SUB (HL)	RES 2,(HL)	
151	SUB A	RES 2,A	
152	SBC A,B	RES 3,B	
153	SBC A,C	RES 3,C	
154	SBC A,D	RES 3,D	
155	SBC A,E	RES 3,E	
156	SBC A,H	RES 3,H	
157	SBC A,L	RES 3,L	
158	SBC A,(HL)	RES 3,(HL)	
159	SBC A,A	RES 3,A	
160	AND B	RES 4,B	LDI
161	AND C	RES 4,C	CPI
162	AND D	RES 4,D	INI
163	AND E	RES 4,E	OUTI
164	AND H	RES 4,H	
165	AND L	RES 4,L	
166	AND (HL)	RES 4,(HL)	
167	AND A	RES 4,A	
168	XOR B	RES 5,B	LDD
169	XOR C	RES 5,C	CPD
170	XOR D	RES 5,D	IND
171	XOR E	RES 5,E	OUTD
172	XOR H	RES 5,H	
173	XOR L	RES 5,L	
174	XOR (HL)	RES 5,(HL)	
175	XOR A	RES 5,A	
176	OR B	RES 6,B	LDIR
177	OR C	RES 6,C	CPIR
178	OR D	RES 6,D	INIR
179	OR E	RES 6,E	OTIR
180	OR H	RES 6,H	
181	OR L	RES 6,L	
182	OR (HL)	RES 6,(HL)	
183	OR A	RES 6,A	
184	CP B	RES 7,B	LDDR
185	CP C	RES 7,C	CPDR
186	CP D	RES 7,D	INDR
187	CP E	RES 7,E	OTDR
188	CP H	RES 7,H	
189	CP L	RES 7,L	
190	CP (HL)	RES 7,(HL)	
191	CP A	RES 7,A	

		nach CB hex	nach ED hex
192	RET NZ	SET 0,B	
193	POP BC	SET 0,C	
194	JP NZ,dddd	SET 0,D	
195	JP dddd	SET 0,E	
196	CALL NZ,dddd	SET 0,H	
197	PUSH BC	SET 0,L	
198	ADD A,dd	SET 0,(HL)	
199	RST 0	SET 0,A	
200	RET Z	SET 1,B	
201	RET	SET 1,C	
202	JP Z,dddd	SET 1,D	
203		SET 1,E	
204	CALL Z,dddd	SET 1,H	
205	CALL dddd	SET 1,L	
206	ADC A,dd	SET 1,(HL)	
207	RST 8	SET 1,A	
208	RET NC	SET 2,B	
209	POP DE	SET 2,C	
210	JP NC,dddd	SET 2,D	
211	OUT dd,A	SET 2,E	
212	CALL NC,dddd	SET 2,H	
213	PUSH DE	SET 2,L	
214	SUB dd	SET 2,(HL)	
215	RST 16	SET 2,A	
216	RET C	SET 3,B	
217	EXX	SET 3,C	
218	JP C,dddd	SET 3,D	
219	IN A,dd	SET 3,E	
220	CALL C,dddd	SET 3,H	
221		SET 3,L	
222	SBC A,dd	SET 3,(HL)	
223	RST 24	SET 3,A	
224	RET PO	SET 4,B	
225	POP HL	SET 4,C	
226	JP PO,dddd	SET 4,D	
227	EX (SP),HL	SET 4,E	
228	CALL PO,dddd	SET 4,H	
229	PUSH HL	SET 4,L	
230	AND dd	SET 4,(HL)	
231	RST 32	SET 4,A	
232	RET PE	SET 5,B	
233	JP (HL)	SET 5,C	
234	JP PE,dddd	SET 5,D	
235	EX DE,HL	SET 5,E	
236	CALL PE,dddd	SET 5,H	
237		SET 5,L	
238	XOR N	SET 5,(HL)	
239	RST 40	SET 5,A	
240	RET P	SET 6,B	
241	POP AF	SET 6,C	
242	JP P,dddd	SET 6,D	
243	DI	SET 6,E	
244	CALL P,dddd	SET 6,H	
245	PUSH AF	SET 6,L	
246	OR N	SET 6,(HL)	
247	RST 48	SET 6,A	
248	RET M	SET 7,B	
249	LD SP,HL	SET 7,C	
250	JP M,dddd	SET 7,D	
251	EI	SET 7,E	
252	CALL M,dddd	SET 7,H	
253		SET 7,L	
254	CP dd	SET 7,(HL)	
255	RST 56	SET 7,A	

* INDEXADRESSIERTE BEFEHLE *

Um die Befehle auf das IY-Register zu beziehen, muss 'DD' in 'FD' geaendert werden.

```
DD 09    ADD IX,BC
DD 19    ADD IX,DE
DD 21    LD IX,dddd
DD 22    LD (dddd),IX
DD 23    INC IX
DD 29    ADD IX,IX
DD 2A    LD IX,(dddd)
DD 2B    DEC IX
DD 34    INC (IX+dd)
DD 35    DEC (IX+dd)
DD 36    LD (IX+dd),dd
DD 39    ADD IX,SP
DD 46    LD B,(IX+dd)
DD 4E    LD C,(IX+dd)
DD 56    LD D,(IX+dd)
DD 5E    LD E,(IX+dd)
DD 66    LD H,(IX+dd)
DD 6E    LD L,(IX+dd)
DD 70    LD (IX+dd),B
DD 71    LD (IX+dd),C
DD 72    LD (IX+dd),D
DD 73    LD (IX+dd),E
DD 74    LD (IX+dd),H
DD 75    LD (IX+dd),L
DD 77    LD (IX+dd),A
DD 7E    LD A,(IX+dd)
DD 86    ADD A,(IX+dd)
DD 8E    ADC A,(IX+dd)
DD 96    SUB (IX+dd)
DD 9E    SBC A,(IX+dd)
DD A6    AND (IX+dd)
DD AE    XOR (IX+dd)
DD B6    OR (IX+dd)
DD BE    CP (IX+dd)
```

```
DD CB 06    RLC (IX+dd)
DD CB 0E    RRC (IX+dd)
DD CB 16    RL (IX+dd)
DD CB 1E    RR (IX+dd)
DD CB 26    SLA (IX+dd)
DD CB 2E    SRA (IX+dd)
DD CB 3E    SRL (IX+dd)
DD CB 46    BIT 0,(IX+dd)
DD CB 4E    BIT 1,(IX+dd)
DD CB 56    BIT 2,(IX+dd)
DD CB 5E    BIT 3,(IX+dd)
DD CB 66    BIT 4,(IX+dd)
DD CB 6E    BIT 5,(IX+dd)
DD CB 76    BIT 6,(IX+dd)
DD CB 7E    BIT 7,(IX+dd)
DD CB 86    RES 0,(IX+dd)
DD CB 8E    RES 1,(IX+dd)
DD CB 96    RES 2,(IX+dd)
DD CB 9E    RES 3,(IX+dd)
DD CB A6    RES 4,(IX+dd)
DD CB AE    RES 5,(IX+dd)
DD CB B6    RES 6,(IX+dd)
DD CB BE    RES 7,(IX+dd)
DD CB C6    SET 0,(IX+dd)
DD CB CE    SET 1,(IX+dd)
DD CB D6    SET 2,(IX+dd)
DD CB DE    SET 3,(IX+dd)
DD CB E6    SET 4,(IX+dd)
DD CB EE    SET 5,(IX+dd)
DD CB F6    SET 6,(IX+dd)
DD CB FE    SET 7,(IX+dd)
DD E1    POP IX
DD E3    EX (SP),IX
DD E5    PUSH IX
DD E9    JP (IX)
DD F9    LD SP,IX
```